



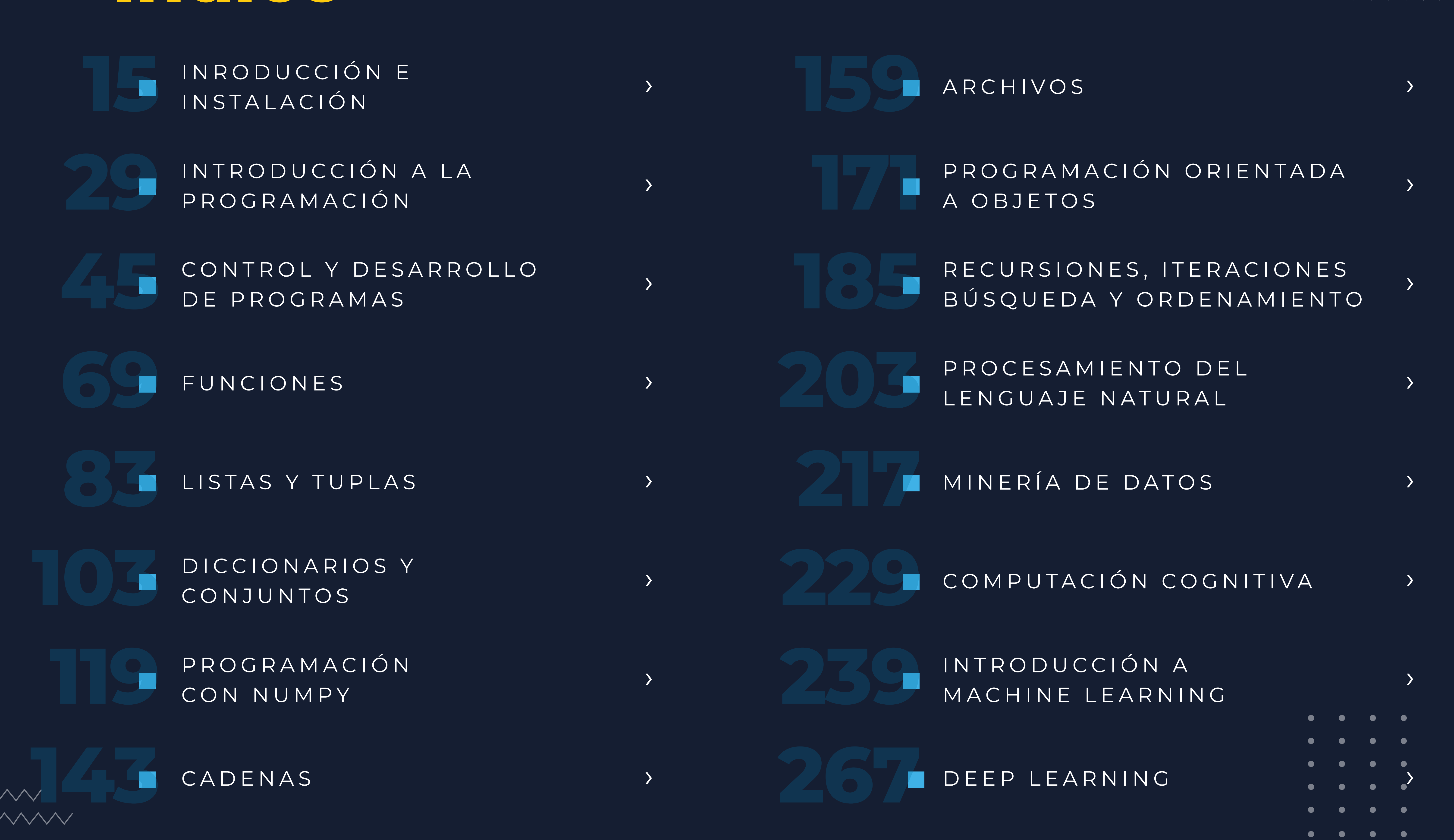


A2CAPACITACION.COM





Índice



15 INRODUCCIÓN E INSTALACIÓN
29 INTRODUCCIÓN A LA PROGRAMACIÓN
45 CONTROL Y DESARROLLO DE PROGRAMAS
69 FUNCIONES
83 LISTAS Y TUPLAS
103 DICCIONARIOS Y CONJUNTOS
119 PROGRAMACIÓN CON NUMPY
143 CADENAS
159 ARCHIVOS
171 PROGRAMACIÓN ORIENTADA A OBJETOS
185 RECURSIONES, ITERACIONES BÚSQUEDA Y ORDENAMIENTO
203 PROCESAMIENTO DEL LENGUAJE NATURAL
217 MINERÍA DE DATOS
229 COMPUTACIÓN COGNITIVA
239 INTRODUCCIÓN A MACHINE LEARNING
267 DEEP LEARNING

Primera edición

Introducción a Python

Manual para el alumno



Introducción a Python

De A2 Capacitación

Guadalajara, Jalisco

Ontario Data Science Consultants Limited

Marzo 2021. Primera Edición

Editor:

Dan Contreras

Diseño:

Samantha Cardenas Reus

Danely Legorreta Parra

Para más detalles ve a: https://a2capacitacion.com/

Tabla de Contenido

Capítulo1. Introducción e instalación de Python.
1.1 Introducción 15
1.2 Acerca de Python 15
1.3 ¿Por qué Python para ciencia de datos? 16
1.4 Instalación de Python 16
1.5 Primero cálculos con Anaconda Prompt 18
1.6 El ambiente de Jupyter Notebook para escribir y ejecutar código 20
1.7 Creando el primer notebook con Jupyter 20
1.8 Ejercicios 25

Capítulo2. Introducción a la programación con Python.
2.1 Introducción 29
2.2 Operadores Aritméticos 29
2.3 Función print, arreglos con comillas simples y dobles comillas 32
2.4 Primeros programas en Python 35
Ejemplo 2.1 Declaraciones condicionales con If 36
Notas 38
Ejemplo 2.2 Valor Máximo de un conjunto de datos 40
2.5 Ejercicios. 41

Capítulo 3. Control y desarrollo de programas con Phyton.
3.1 Introducción 45
3.2 Detrás del telón de la programación, Algoritmos, Pseudocódigo y otras monadas 45
3.3 Algoritmos 46
3.4 Pseudocódigo 46
3.5 Sentencias de Control 47
3.6 Sentencias de decisión y repetición 47
3.7 Sentencia if 48
3.8 Sentencias if…else, if…elif…else 49
Nota 50
3.9 Sentencia While 54
3.10 Sentencia for 55
Ejemplo 3.1 57
Ejemplo 2 59
Ejemplo 3 60
Notas 63
3.11 Operadores Booleanos 65
Operador Booleano and 65
Operador Booleano or 66
3. 12 Ejercicios 66

Capítulo 4. Funciones.
4. 1 Introducción 69
4.2 Función def 69
Notas 70
4.3 Funciones con Múltiples Parámetros 71
4.4 Números aleatorios 71
Ejemplo 1 72
4.5 Funciones sin parámetros y múltiples parámetros 75
4.6 Alcance local y global 77
4.7 El módulo de matemáticas de Python 78
4.8 Medidas de Tendencia Central y de Dispersión 79

Capítulo 5. Lista y Tuplas.
5.1 Introducción 83
5.2 Listas 83
5.3 Tuplas 86
5.4 Separando secuencias 88
5.5 Listas ordenadas 90
5.6 Comprensión de listas 93
5.7 Listas de 2 dimensiones 98
5.8 Visualización de datos estáticos 99

Capítulo 6. Diccionarios y conjuntos.
6.1 Introducción 103
6.2 Dicccionarios 103
Ejemplo 1 107
Ejemplo 2 108
6.3 Comprensión de diccionarios 111
6.4 Conjuntos (Sets) 111
Unión de conjuntos. 113
Intersección de conjuntos 114
Diferencia de conjuntos 114
Diferencia simétrica de conjuntos 114
Conjuntos disjuntos 115

Capítulo 7. Programación con Numpy.
7.1 Introducción 119
7.2 Arreglos de Datos 119
7.3 Operaciones con arreglos 123
7.4 Indexando y cortando arreglos 126
7.5 Pandas 131
7.6 Dataframes 134

Capítulo 8. Cadenas.
8.1 Introducción 143
8.2 Dando formato a las cadenas. 143
8.3 Otras opciones para las cadenas 147
8.4 Expresiones Regulares 152
8.5 Pandas y Expresiones Regulares 156

Capítulo 9. Archivos.
9.1 Introducción 159
9.2 Archivos TXT 159
Notas 160
9.3 Archivos JSON 162
9.4 Manipulación de Excepciones 164
Ejemplo 1 165
9.5 Archivos CSV 166
9.6 Lectura de archivos CSV con DataFrame de Pandas 167

Capítulo 10. Programación orientada a objetos.
10.1 Introducción 171
10.2 Métodos y Clases 171
10.3 La clase Cuenta 175
10.4 Herencia 176
10.5 La clase Personal 176
10.6 Métodos de clase 179
10.7 Polimorfismo 180

Capítulo 11. Recursiones, iteraciones, búsqueda y ordenamiento.
11.1 Introducción 185
11.2 Algoritmos Recursivos e Iterativos 185
Ejemplo 1 186
Ejemplo 2 187
Ejemplo 3 188
11.3 Búsqueda 190
11.4 Ordenamiento 193

Capítulo 12. Procesamiento de Lenguaje Natural.
12.1 Introducción 203
12.2 Instalación del módulo TextBlob 203
12.3 TextBlob 204
12.4 Detección de lenguaje y traducción 207
12.5 Visualizando frecuencias de palabras 212

Capítulo 13. Minería de datos.
13.1 Introducción 217
13.2 Minería de Datos 218
13.3 Etapa 1. Extracción y limpieza de la información 219
13.4 Parte 2. Preprocesamiento de la información 220
13.5 Parte 3. Análisis y visualización de los datos 222

Capítulo 14. Computación cognitiva.
14.1 Introducción 229
14.2 Servicios de Watson de IBM 230
14.3 Ejemplo de un traductor básico 232

Capítulo 15. Introducción a Machine learning.
15.1 Introducción 239
15.2 Tipos de Machine Learning 239
15.3 Machine Learning Supervisado. 240
15.4 Cargar la base de datos 242
15.5 Transformación de datos 243
15.6 Exploración básica de datos 244
15.7 Separación de datos para Entrenamiento y Prueba 245
15.8 Creación del modelo 246
15.9 Entrenamiento y prueba del modelo 246
15.10 Pronóstico de clases de dígitos 247
15.11 Ajuste y evaluación del modelo 247
15.12 Validación Cruzada K-Fold 250
15.13 Regresión Lineal Múltiple 252
15.14 Cargar la base de datos 253
15.15 Exploración básica de datos 253
15.16 Separación de datos para entrenamiento y prueba 259
15.17 Creación, entrenamiento del modelo 259
15.18 Prueba del modelo 261
15.19 Ajuste y evaluación del modelo 262
15.20 Comparando diferentes modelos 262
15.20 Machine Learning No Supervisado. 263

Capítulo 16. Aprendizaje profundo (Deep Learning)
16.1 Introducción 267
16.2 Keras 267
16.3 Instalación de VectorFlow y Keras 269
16.4 Redes Neuronales 269
16.5 Capas convolucionales 271
16.6 Tensores 273
16.7 Clasificación Múltiple 274
16.8 Carga de Datos 274
16.9 Exploración de datos 274
16.10 Preparación de los datos 276
16.11 Entrenamiento y Evaluación del Modelo 279
16.12 Análisis de Emociones con Keras 283
16.13 Carga de Datos 283
16.14 Exploración de Datos 284
16.15 Preparación de los datos 286
16.16 Creación de la red neuronal 287

CAPÍTULO 1

Introducción e Instalación de Python

1.1 Introducción

En este capítulo es para compartirte algunas de las causas que hacen de Python la mejor herramienta de programación para Ciencia de Datos. Conoce un poco de las bondades que tiene este software, del mundo de aplicaciones donde lo puedes encontrar y de las grandes oportunidades que se te pueden abrir al aprender a programar en este lenguaje.

También aprenderás la forma en que lo puedes instalar en tu computadora a través de Anaconda. Anaconda además de proporcionar un intérprete de Python, contiene otros ambientes que utilizaremos en este manual, como Jupyter Notebook y JupyterLab.

"Así que sin más preámbulo, ... iniciemos la aventura"

1.2 Acerca de Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma. *(https://es.wikipedia.org/wiki/Python)*

Algunas ventajas con el uso de Python:

- **Es fácil de usar.** Con algunos tips sencillos, cualquier persona puede programar rutinas sencillas en Python. Pronto te darás cuenta de ello.
- **Legibilidad del código.** La estructura del código es bastante natural y promueve una forma de escribir que facilita su lectura. Esta es una ventaja importante frente a lenguajes dirigidos al mismo sector, como Perl.
- **Facilidad de uso en dispositivos.** Algunas plataformas como Raspberry y Numpy están basadas en Python.

- **Facilidad de escritura de código asíncrono.** Los lenguajes diseñados antes de que las plataformas multiprocesador (o multinúcleo) estuvieran tan generalizadas suelen tener estructuras bastante complicadas para mantener distintos hilos de ejecución; en Python el código asíncrono es bastante sencillo de gestionar.
- **Abundancia de bibliotecas.** Hay muchas bibliotecas disponibles para extender la funcionalidad básica de Python a cualquier campo.
- **Gran base de usuarios.** Esto hace que exista mucho código disponible en internet y que los foros de usuarios sean bastante activos, por lo que es fácil encontrar ayuda cuando se necesita.

Python, naturalmente, tiene también sus desventajas, algunas de las cuales son consecuencia de sus ventajas. Por ejemplo: la facilidad de uso tiene como contrapartida una menor flexibilidad o rapidez de ejecución que otros lenguajes como el mismo C, que sigue siendo el líder en programación de bajo nivel o cuando la rapidez de ejecución es crítica. Y también tiene algunos problemas de seguridad, lo que es crucial en internet. Ningún lenguaje es la panacea para todas las aplicaciones.

Desde <https://es.quora.com/Qu%C3%A9-ventajas-nos-ofrece-Python-respecto-a-otros-lenguajes-de-programaci%C3%B3n>

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”

1.3 ¿Por qué Python para ciencia de datos?

Python es un lenguaje que todo el mundo debería conocer. Su sintaxis simple, clara y sencilla; el tipado dinámico, el gestor de memoria, la gran cantidad de librerías disponibles y la potencia del lenguaje, entre otros, hacen que desarrollar una aplicación en Python sea sencillo, muy rápido y, lo que es más importante, divertido.

La sintaxis de Python es tan sencilla y cercana al lenguaje natural que los programas elaborados en Python parecen pseudocódigo. Por este motivo se trata además de uno de los mejores lenguajes para comenzar a programar.

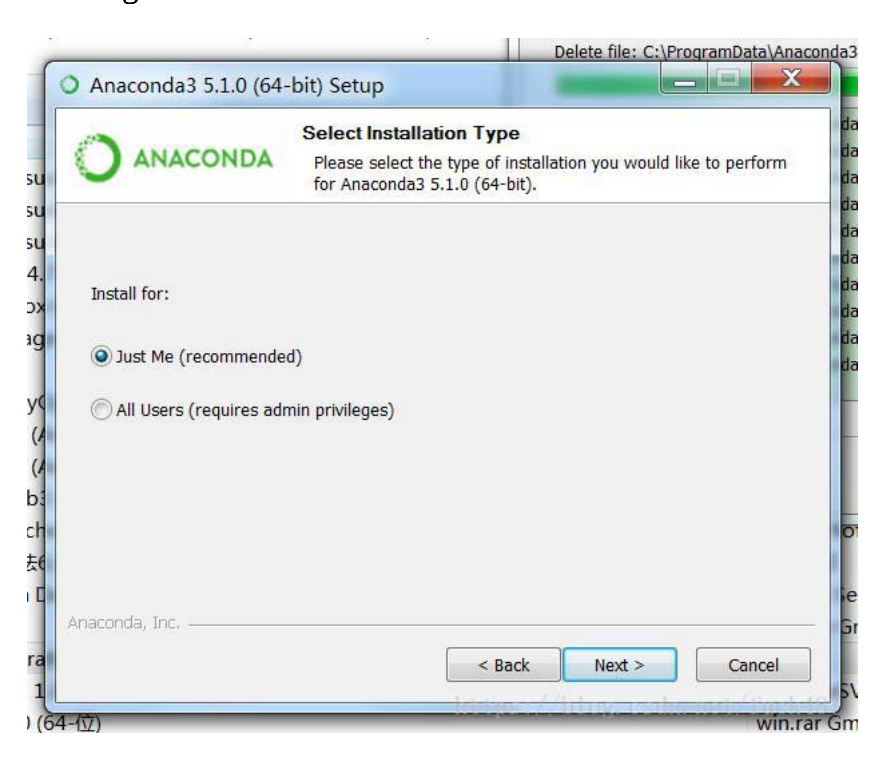
Algunos casos de éxito en el uso de Python son Google, Yahoo, la NASA, Industrias Light & Magic, y todas las distribuciones Linux, en las que Python cada vez representa un tanto por ciento mayor de los programas disponibles. (González, R. ())

1.4 Instalación de Python

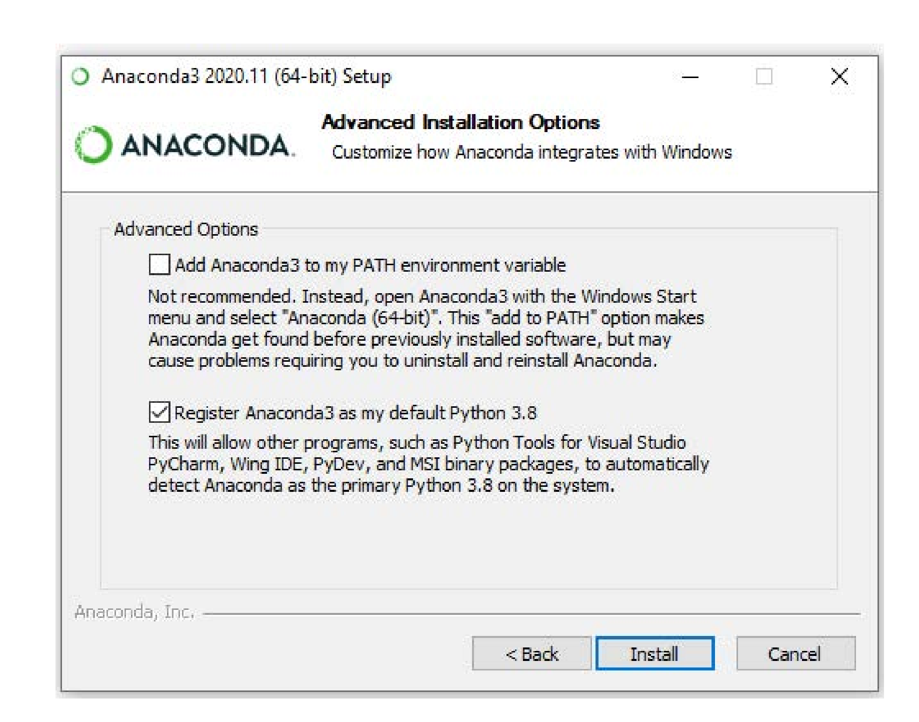
Para este curso utilizaremos el ambiente generado por Anaconda para programación en Python. La descarga puede realizarse desde <*https://www.anaconda.com/products/individual*>
Selecciona el archivo (instalador) adecuado de acuerdo al sistema operativo que tiene tu computadora (Windows, Mac o Linux)

Al finalizar la descarga ya puedes continuar con la instalación de Anaconda.

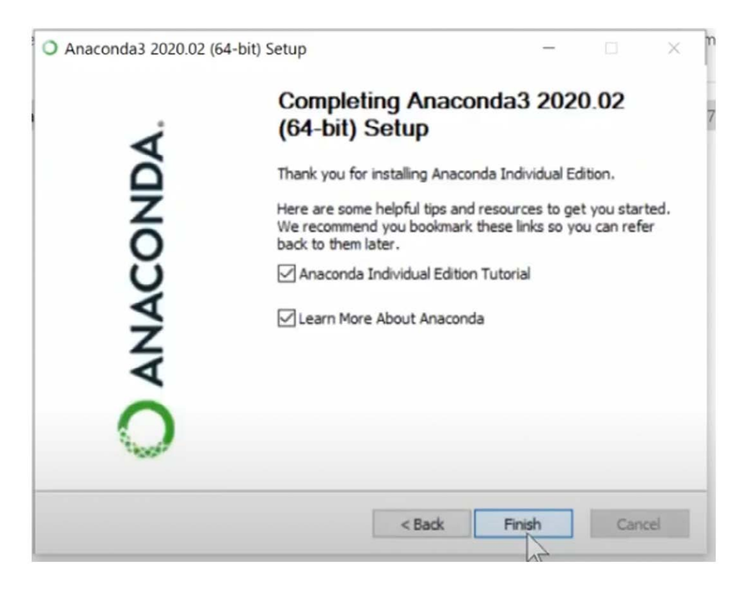
Regularmente la instalación es rápida y muy fácil. Un par de recomendaciones, cuando salga el siguiente cuadro de diálogo:



Deja la opción que se encuentra ya seleccionada. Cambiar a All Users, puede provocar conflictos posteriores con los permisos de instalación de algunas librerías. Verás que también aparecerá el cuadro:



Esta es la selección recomendada para las opciones avanzadas. Continua con la instalación hasta llegar al final.

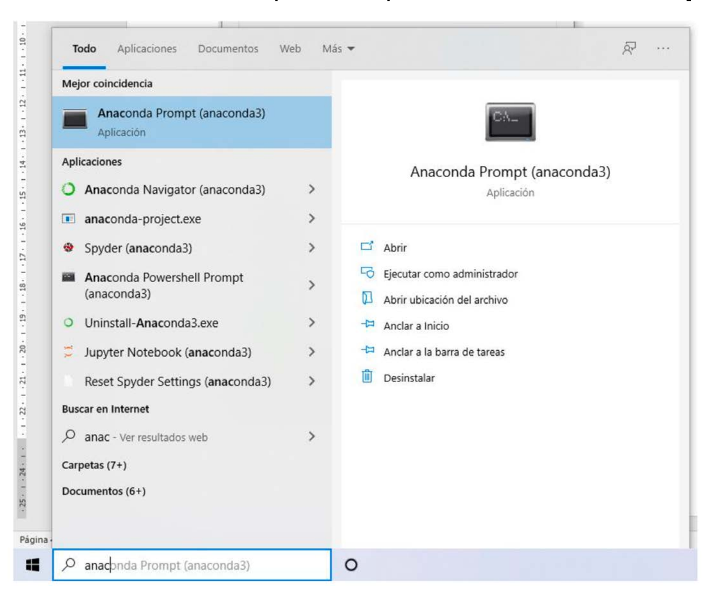


Con la instalación de Anaconda, se han incluido:

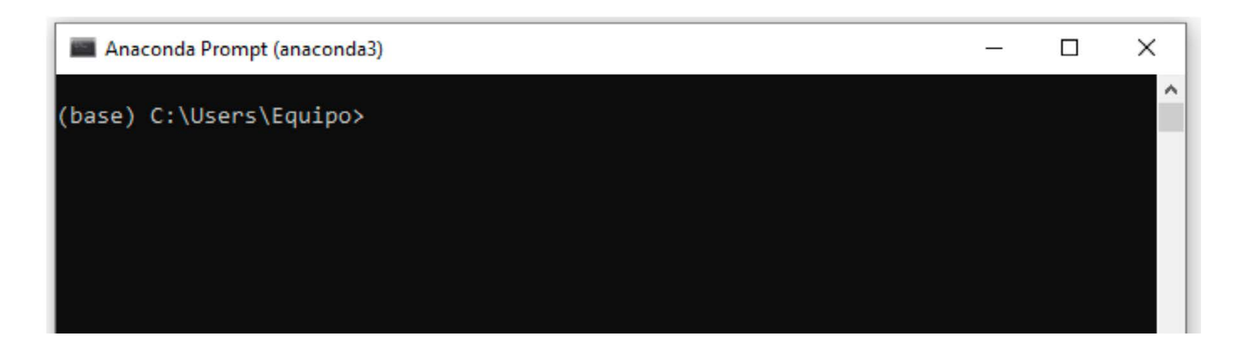
- El interprete IPython
- La mayoría de las librerías de Ciencias de Datos que utilizaremos
- Un servidor local llamado Jupyter Notebooks, donde ejecutarás las notas (apuntes, cálculos, imágenes, etc.) que te permitirán aprender a programar en Python.
- Otras aplicaciones, como Spyder, Orange 3, PyCharm Community, entre otros.

1.5 Primero cálculos con Anaconda Prompt

Para probar que la instalación ha sido bien realizada, haremos algunas pruebas con el prompt de Anaconda. Para acceder, primero abre Anaconda Prompt. Desde Windows lo puedes hacer desde el menú de inicio, o escribiendo en la búsqueda rápida **Anaconda Prompt**



Al abrir el Anaconda Prompt aparece una ventana como la siguiente



Teclea, ipython, y presiona Enter (o Return). Verás algo como esto

```
IPython: C:Users/Equipo

(base) C:\Users\Equipo>ipython
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.19.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

El texto **"In[1]:"** es llamado prompt, éste, es un indicador de que el interprete IPython está esperando tus indicaciones. Al trabajar en este ambiente se le llama **modo interactivo** y las sentencias cortas que se pueden escribir en código Python se les llama **snippets.**

En el modo interactivo es posible evaluar expresiones sencillas, por ejemplo

```
In[1]: 89 - 20
Out[1]: 69
```

Después de que escribes `89-20` y presionas Enter, IPython lee el snippet, lo evalúa e imprime el resultado en `Out[1]`. Observa que IPython muestra ahora el prompt `In[2]` para indicar que está en espera de un segundo snippet. Esto lo realizará recurrentemente para cada snippet nuevo que agregues.

Ejemplos:

- Empleando el modo interactivo, calcula la suma 3+4*5

```
In[2]: 3 + 2 * 5
Out[2]: 13
```

- Ahora calcula (3+2)*5

```
In[3]: (3 + 2) * 5
Out[3]: 25
```

En el primer ejemplo, observa que hay prioridad en las operaciones. Ya que, en programación, la multiplicación es una operación que tiene prioridad sobre la suma.

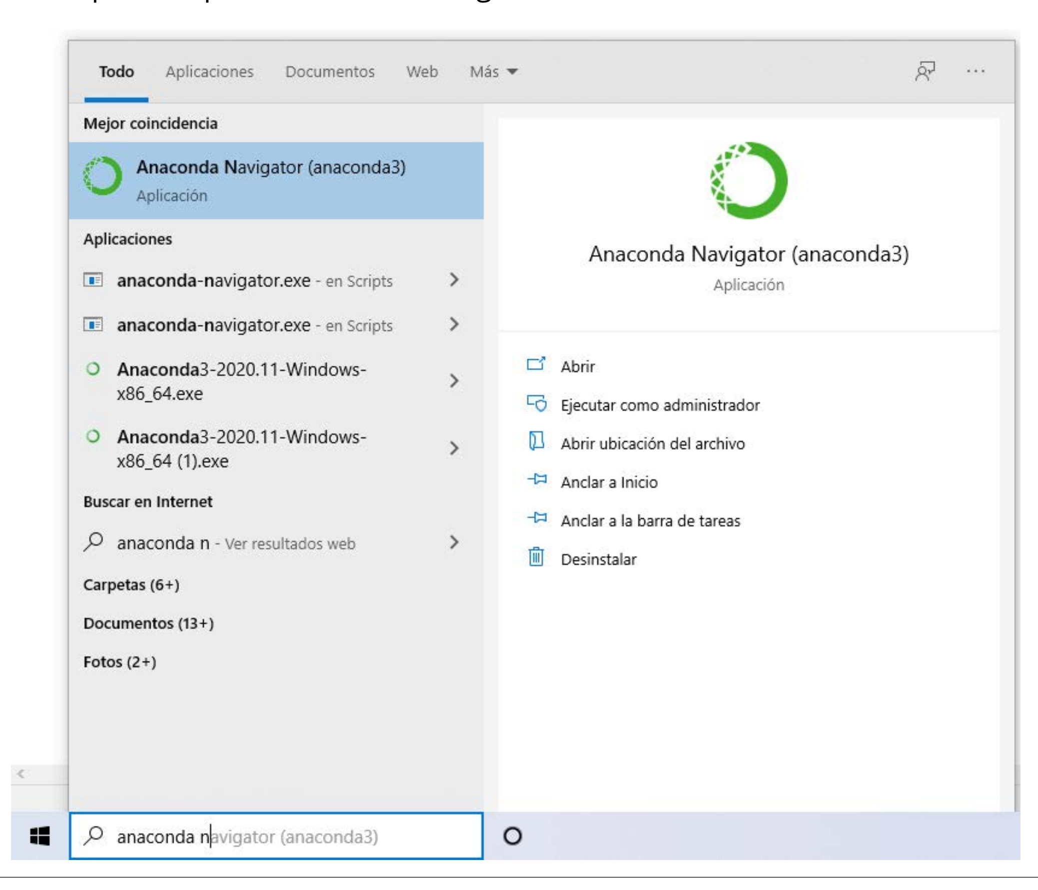
En el segundo ejemplo, observa que el paréntesis tiene mayor prioridad que la multiplicación. Justo como en los cursos de matemáticas.

1.6 El ambiente de Jupyter Notebook para escribir y ejecutar código

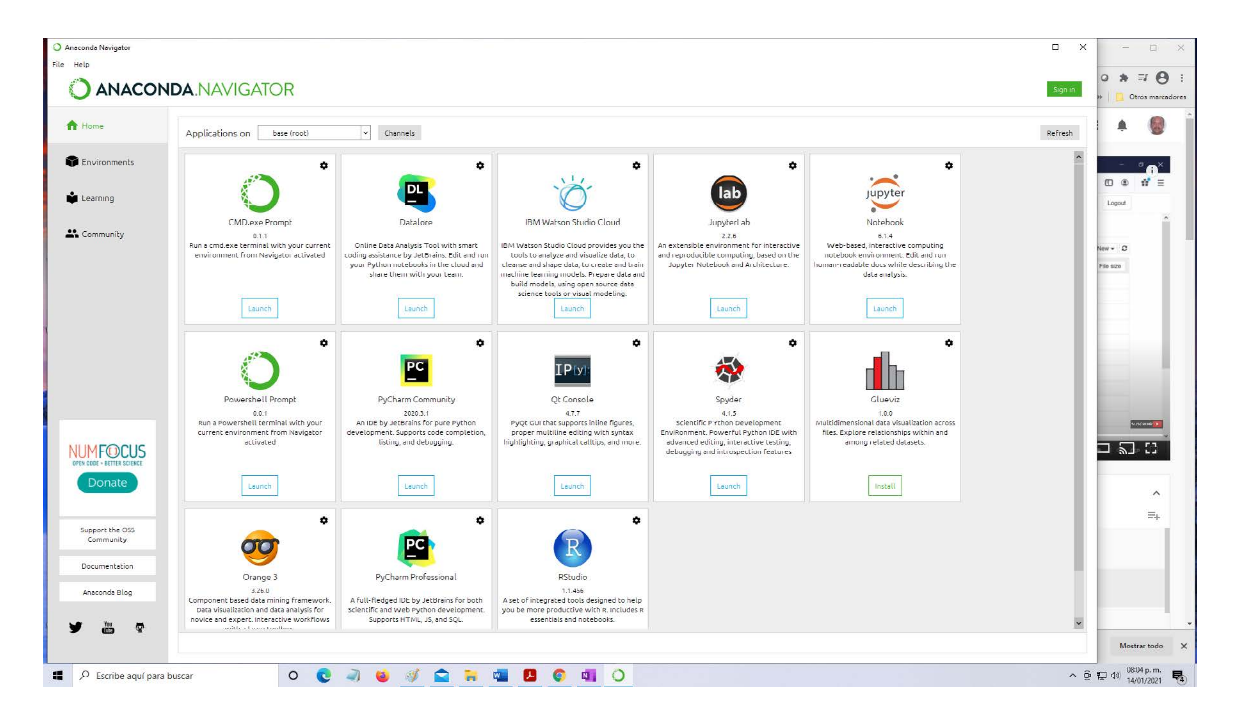
Una de las grandes ventajas de instalar Anaconda es la instalación simultánea de otras herramientas, en este caso utilizaremos Jupyter Notebook y JupyterLab. Ambos trabajan en un entorno de página Web en donde es posible escribir y ejecutar código, además de mezclar texto, imágenes y video. Esta versatilidad permite compartir muchos resultados basados en Python.

1.7 Creando el primer notebook con Jupyter

Para acceder al Jupyter Notebook, en Windows, lo puedes hacer desde el menú de inicio, o escribiendo en la búsqueda rápida Anaconda Navigator

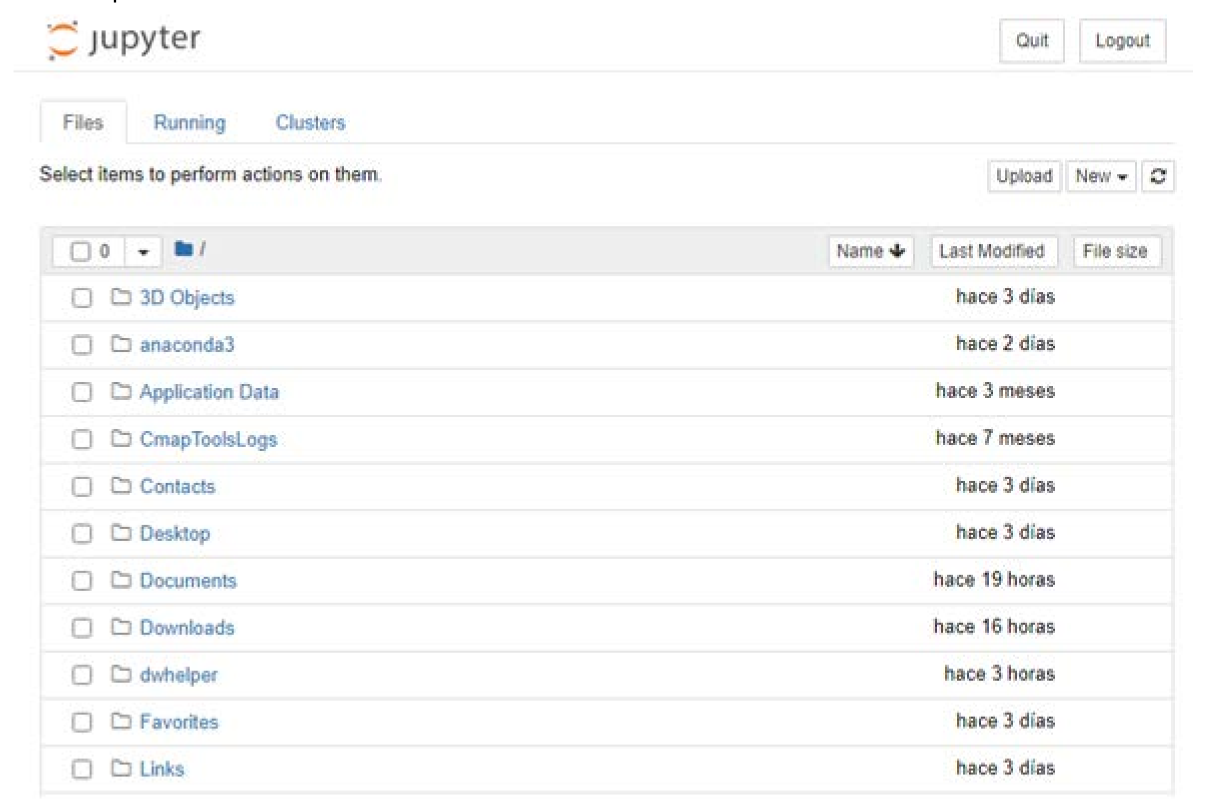


Cuando se carga Anaconda, podemos ver las aplicaciones que contiene

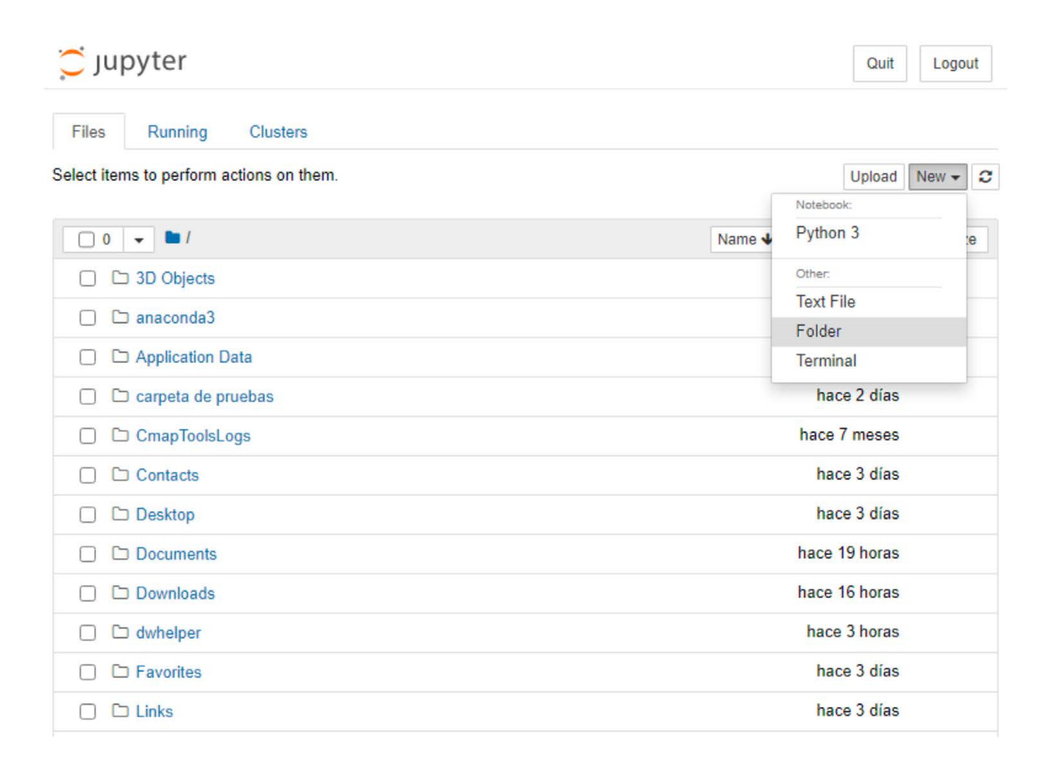


Inicia el ambiente Jupyter Notebook. En el navegador predeterminado de tu computadora se abrirá una pestaña como la que se muestra.

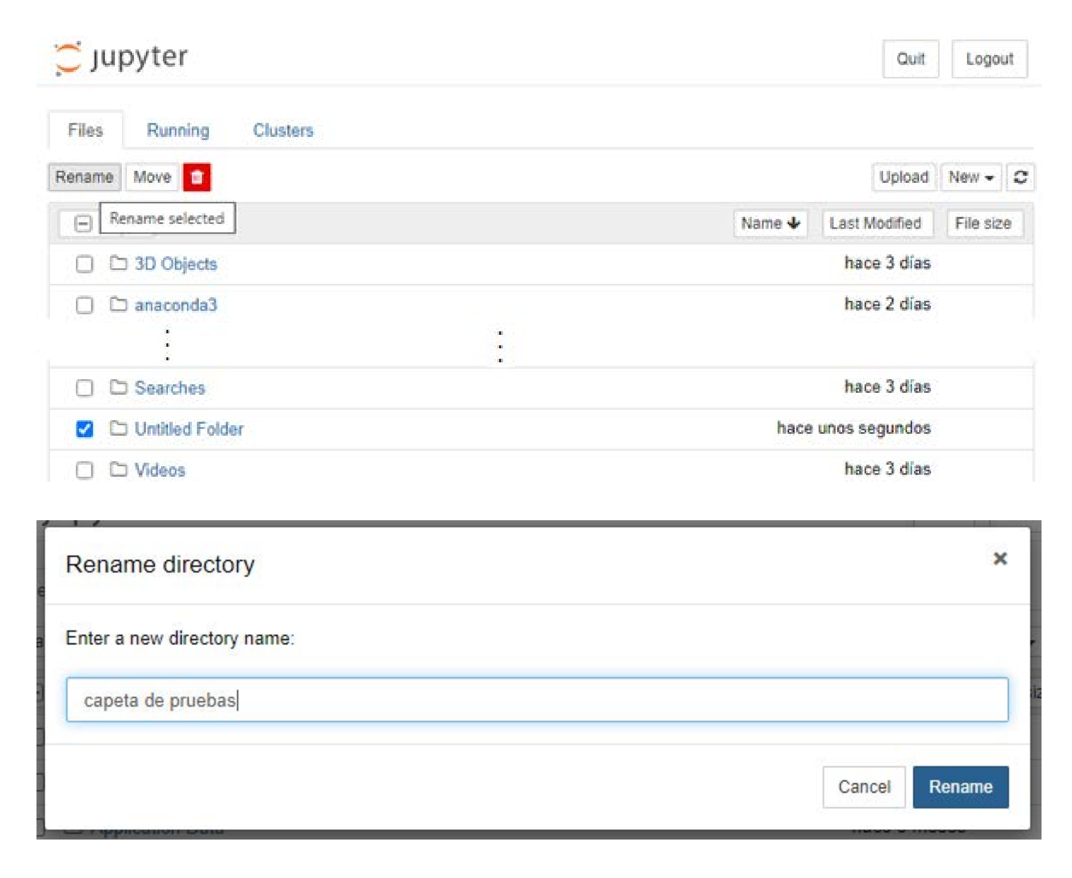
Crea una carpeta nueva.



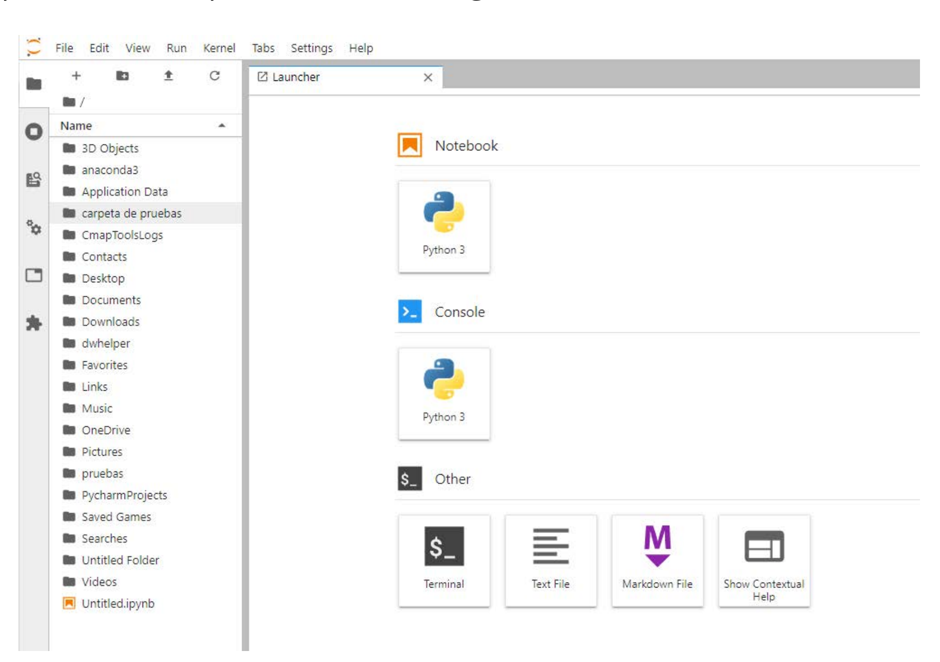
La carpeta se crea con el título Untitled Folder, selecciónala y cambia su nombre por: carpeta de pruebas.



Una vez que la carpeta está creada, accede a JupyterLab.

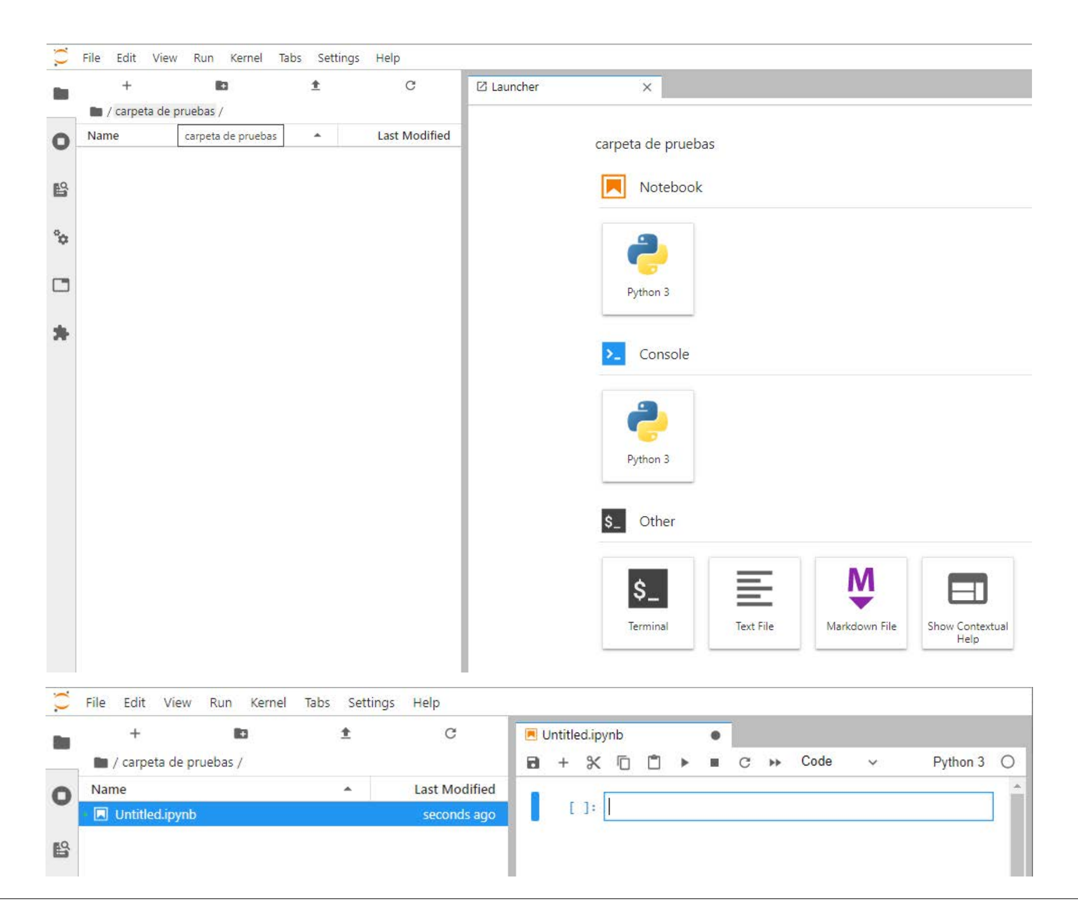


JupyterLab aparece en otra pestaña de tu navegador.



En la ventana de JupyterLab aparece un menú con las carpetas que se muestra en el Notebook de Jupyter. Observa que también aparece la carpeta que creamos.

Accede a ella verás que está vacía.Aquí selecciona un Notebook de Python 3.

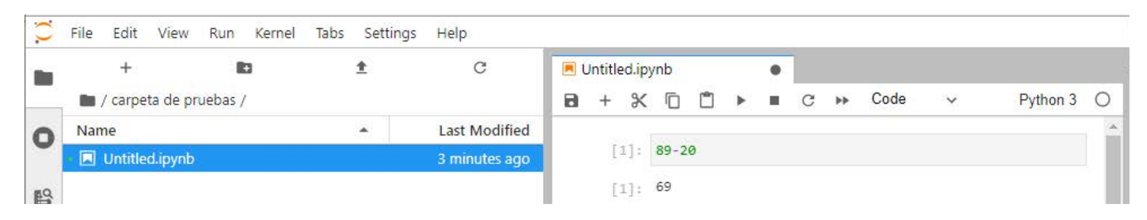


La unidad de trabajo dentro del notebook es una **celda** (cell) en donde puedes escribir un snipett. Por default, un notebook nuevo contiene una celda. Es posible añadir más celdas presionando el ícono +, del menú de herramientas.

Da click en la celda y escribe la expresión 89-20

Para ejecutar el código de la celda puedes presionar el ícono ▶ , o con el teclado Ctrl+Enter.

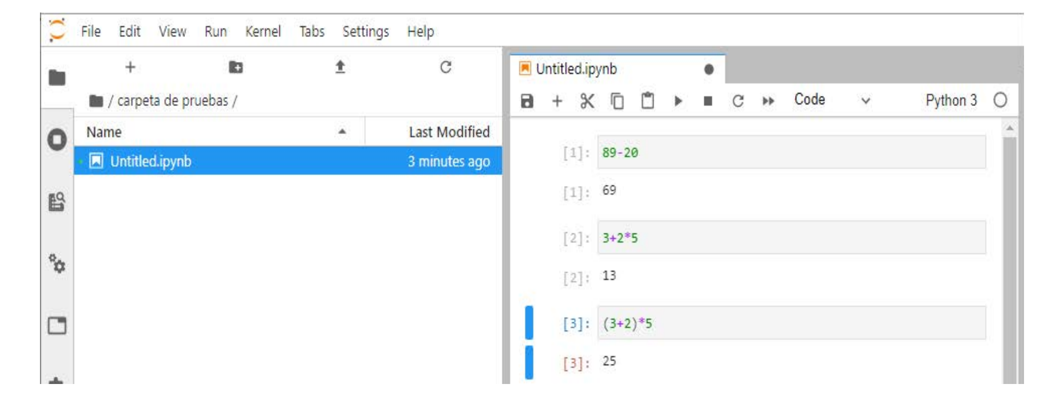
JupyterLab ejecuta el código en IPython y muestra el resultado



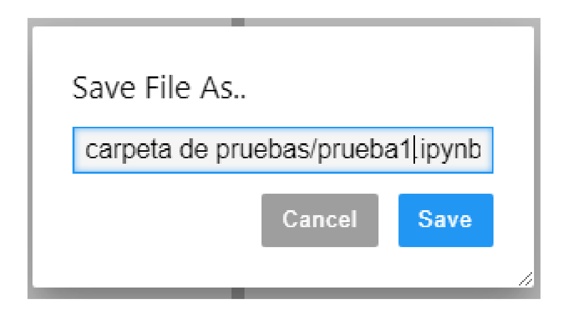
Verás que ya aparece otra celda nueva. Si requieres más celdas, presiona cuando las requieras agregar.

Evalúa las otras expresiones que habías realizado con Anaconda Prompt.

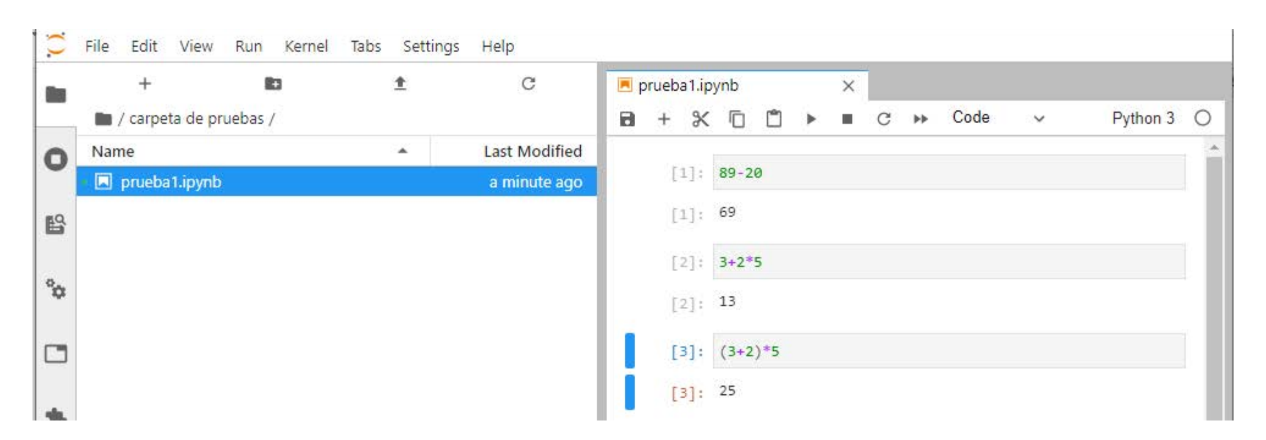
Escribe 3+2*5, y ejecuta la celda. Para finalizar lo ejercicios, escribe (3+2)*5 en una nueva celda y verifica que JupyterLab devuelve los mismos resultados.



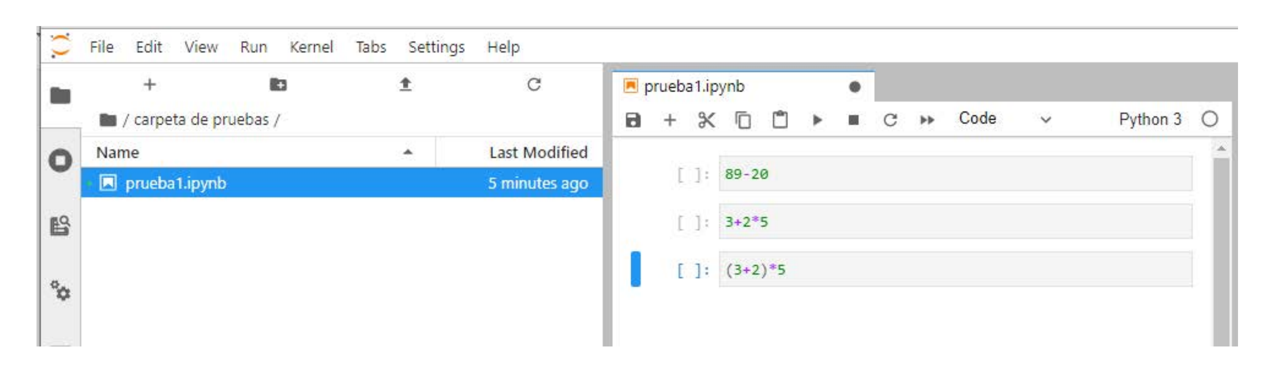
Si el Notebook tiene cambios que no has guardado, la × en el notebook para cerrar JupiterLab cambiará a ● . Para guardar el notebook selecciona el ícono 💾, o bien, en el menú de JL selecciona File y luego **Save Notebook**. Escribe el nombre **prueba1**.



Guarda el archivo (Save).



En los materiales que acompañan a este manual se encuentran los notebooks con todos los ejemplos y ejercicios que realizarás. Estos notebooks no tienen ejecutadas las celdas. Si las ejecutas, pero las quieres regresar al código original, del menú del *Kernel* selecciona *Restart Kernel* y luego *Clear All Outputs*. El notebook quedará así



Vuelve a guardar el notebook para cerrarlo.

1.8 Ejercicios

1. En la división aritmética, por ejemplo 17/6, el cociente es 2 y el residuo es 5. De tal manera que

$$6*2+5=17$$

En Phyton, el operador que muestra el cociente (parte entera de la división 17/6) es //. Y el operador que calcula el residuo (resto, o módulo) es %. Utiliza el prompt de Anaconda para determinar si los siguientes enunciados son verdaderos o falsos.

- **a.** 8 // 3 =2
- **b.** 8 % 3= 1
- **c.** 111 // 6=16
- **d.** 194 % 2=0
- **e.** La parte entera de 326 entre 15 es, 22
- **f.** El residuo de dividir 978 entre 125 es, 103
- **g.** 25 módulo 3 es, 1

2. Utiliza **Anaconda Prompt** para realizar los siguientes cálculos. Identifica aquel que genera un error y explica la razón por la que ocurre.

   a. 25 / 4=
   b. 25 // 4=
   c. 0 / 4=
   d. 0 // 4=
   e. 25 / 0=
   f. 25 // 0=

3. Utiliza Jupyter Notebook y JupiterLab para realizar los siguientes cálculos. Guarda tus resultados en un archivo con nombre prueba2. Identifica aquel que genera un error y explica la razón por la que ocurre.

   a. 32 / 8 -3=
   b. 32 // 8 -3=
   c. 32 / (8 -3)=
   d. 32 // (8 -3)=
   e. 32 / (8 -8)=
   f. 32 / (8 -5)=

CAPÍTULO 2

Introducción a la programación con Python

2.1 Introducción

En este capítulo aprenderás algunas características de programación en Python mediante algunos ejemplos ilustrativos. La filosofía de este manual es aprender programando, así que:

"Acerca tu bebida preferida, enciende tu computadora, respira profundo y pongámonos a programar"

2.2 Operadores Aritméticos

Ya utilizaste IPython para realizar algunos ejercicios en el modo interactivo. Este modo interactivo lo realizaste con Anaconda Prompt, por ejemplo:

```
In[1]: 9 + 2
Out[1]: 11
```

Los siguientes ejemplos también utilizan Anaconda Prompt, abre esta consola y verifica los resultados que se muestran.

Es un cálculo aritmético de matemáticas. Pero como en matemáticas, algunas expresiones de este tipo pueden tener variables

```
In[2]: w = 39
```

El snippet [2] es una declaración. Cada declaración define una tarea a realizar. En este caso, el snippet crea la variable w a la que le asigna, a través del signo = el valor de 9.

Es útil realizar estas declaraciones, ya que las declaraciones permiten almacenar valores que pueden ser útiles posteriormente. Por ejemplo:

```
In[1]: 9 + 2
Out[1]: 11
In[3]: z = 7 In[4]: w / z
```

Continuando con esta forma de declarar las variables, también podemos asignar

```
In[5]: division = w / z
In[6]: division
Out[6]: 5.571428571428571
```

Observa lo que ocurre si escribes

```
In[7]: Division
<ipython-input-5-75d6e457f232> in <module>
----> 1 Division
NameError: name ‘Division’ is not defined
---------------------------------------------------
NameError Traceback (most recent call last)
```

Aparece un error. Esto ocurre porque Python reconoce mayúsculas y minúsculas, de manera que *"division"* y *"Division"* se identifican como dos cosas diferentes. La primera que está definida en el snippet [5] y la segunda que no está definida de manera previa. Es por esto que Python indica que la variable no está definida.

Otra observación importante es el tipo de número que se muestra en el prompt de Anaconda, para hacer más visible este detalle escribe en el prompt type(w), verás que Python reconoce a la variable w como un entero.

```
In[8]: type(w)
Out[8]: int
```

¿De qué tipo es la variable división?. Teclea type(división)

```
In[9]: type(division)
Out[9]: float
```

Observa que el Python indica que la variable es de tipo float, esto indica que el número tiene una parte decimal. Por default, Python asigna un número tipo float, aunque el resultado en alguna división pueda tener o no, parte decimal.

Seguramente notaste en los ejercicios del capítulo anterior que es posible realizar diversas operaciones aritméticas. La tabla siguiente muestra algunos de los operadores aritméticos más comunes.

	Símbolo	Expresión algebraica	Expresión en Python	Ejemplo
Suma	+	a + b	a + b	In[1]: 9 + 2 Out[1]: 11
Resta	−	a − b	a − b	In[2]: 9 + 2 Out[2]: 7
Multiplicación	*	a * b	a * b	In[3]: 9 * 2 Out[3]: 18
División	/	$\frac{a}{b}$	a / b	In[4]: 9 / 2 Out[4]: 4.5
División parte entera	//	$\left[\frac{a}{b}\right]$	a // b	In[4]: 9 // 2 Out[4]: 4
Módulo	%	a mod b	a % b	In[5]: 8 % 2 Out[5]: 0
Exponente	**	a^b	a ** b	In[6]: 8 ** 2 Out[6]: 64

En matemáticas una operación que no está definida es la división entre cero. Observa que al escribir 3/0 aparece la leyenda

```
In [10]: 3/0
---------------------------------------------------
ZeroDivisionError Traceback (most recent call last)
<ipython-input-1-f6cc6d14333b> in <module>
----> 1 3/0
ZeroDivisionError: division by zero
```

Esta leyenda indica que hay una excepción del tipo división entre cero `(ZeroDivisionError).`

Muchas excepciones son reconocidas como errores.

La línea -->1 indica el lugar en el código que provoca la excepción. La última línea indica la excepción que ocurre

A continuación, se presenta una tabla que muestra el orden o prioridad que tienen las operaciones en Python

1. Paréntesis: ( )
2. Exponente: **
3. Multiplicación, División, División entera, Módulo: *, /, //, %
4. Suma, Resta: +, −

El siguiente ejemplo muestra como Python aplica la prioridad cuando se realizan diferentes operaciones.

```
In[11]: 1+2**4%5
Out[11]: 2
```

Debido a al prioridad de operaciones, Python primero calcula

$$2**4 \equiv 2^4 = 16$$

La triple línea ≡ es para indicar que las expresiones son equivalentes. Luego realiza la operación módulo entre 16 y 5

$$16\%5 \equiv 16\ mod5 = 1$$

Recuerda que la operación módulo es el residuo en una división.

Y la operación final que realiza es la suma

$$1 + 1 = 2$$

Demaneraque $1+2^4 mod5=1+16mod5=1+1=2$

2.3 Función print, arreglos con comillas simples y dobles comillas

Utiliza el prompt de Anaconda y ejecuta el siguiente snippet

```
In[1]: print(‘Hoy aprenderé a usar la función print’)
Hoy aprenderé a usar la función print
```

Ahora ejecuta

```
In[2]: print(“Hoy aprenderé a usar la función print”)
Hoy aprenderé a usar la función print
```

Observa que los resultados son exactamente iguales. Es decir, no hay diferencia el resultado de ambas instrucciones. Así que muchos programadores prefieren utilizar comillas simples. Sin embargo, esto que parece ser una redundancia de Python en realidad tiene una aplicación práctica. Por ejemplo, para declarar el arreglo

Hoy aprenderé a usar la función ‘print’

```
In[3]: print(‘Hoy aprenderé a usar la función ‘print’ ’)
  File "<ipython-input-3-fa3cc41852b0>", line 1
    print('Hoy aprenderé a usar la función 'print'')
SyntaxError: invalid syntax
```

Se produce un error de sintaxis. Esto se resuelve utilizando dobles comillas

```
In[4]: print(“Hoy aprenderé a usar la función ‘print’”)
Hoy aprenderé a usar la función ‘print’
```


Algo que es importante atender, es que no aparecen los resultados de los snippets (Out[1], Out[2])

Esto ocurre porque la función `print` solicita a Python trabajar en un ambiente de editor de texto, en lugar de realizar cálculos, como antes lo habías hecho.

Esta diferencia será más obvia cuando trabajemos líneas de texto en el ambiente de JupyterLab. Otras opciones para insertar texto el prompt de Anaconda son

```
In[5]: print(‘Hoy aprenderé’,’a usar’, ‘la función print’)
Hoy aprenderé a usar la función print
```

```
In[6]: print(‘Hoy aprenderé\n a usar\n la función print’)
Hoy aprenderé
 a usar
 la función print
```

```
In[7]: print(‘También aprenderé a dividir \
…: líneas de texto cuando las expresiones \
…: sean demasiado largas’)
También aprenderé a dividir líneas de texto cuando las ex-
presiones sean demasiado largas
```

El símbolo \ al final de la línea permite continuar escribiendo caracteres en otra línea, ignorando el salto de línea, los símbolos …: se colocan automáticamente después de presionar `enter` en el prompt.

Cuando requieras utilizar comillas dobles y sencillas en algún enunciado, por ejemplo

Aprender ‘Python’, es realmente “sencillo”

Es posible hacerlo utilizando triples comillas

```
In[8]: print(“““Aprender ‘Python’ es realmente “sencillo” ”””)
Aprender ‘Python’ es realmente “sencillo”
```

Con Python, puedes asignar a una variable un arreglo de caracteres,

```
In[9]: arreglo1=‘Hoy aprenderé a usar la función print’
In[10]: print(arreglo1)
Hoy aprenderé a usar la función print
```

Como en el caso de las variables numéricas, las cadenas de caracteres o arreglos tienen un formato específico, que Python lo interpreta como un *string* (cadena de caracteres).

```
In[11]: type(arreglo1)
Out[11]: str
```

En el siguiente cuadro se muestran algunos de los comandos útiles para anexar comillas a un arreglo, saltos de página y espaciado horizontal

	Descripción	Ejemplo
\n	Insertar un salto de línea	`In[]:print('Línea 1\n Línea 2')` `Línea 1` `Línea 2`
\\	Inserta una diagonal invertida (backslash)	`In[]:print('\\')` `/`
\'	Inserta una comilla sencilla	`In[]:print('\'')` `'`
\"	Inserta una comilla doble	`In[]:print('\"')` `"`
\t	Inserta un espacio (tab) horizontal	`In[]:print('Línea 1\n Línea 2')` `Línea 1          Línea 2`

Con estas herramientas, ya puedes mostrar los resultados de los cálculos en una forma más amigable

```
In[12]: w=9
In[13]: z=2
In[14]: potencia=w**z
In[15]: print('Nueve con potencia dos, es',potencia)
Nueve con potencia dos, es 81
```

En algunas situaciones es necesario que el usuario ingrese algún dato, por ejemplo: la edad, la estatura, el nombre, el número telefónico, etc. Aquí la función clave es `input`, que solicita al usuario la información específica que debe ingresar al programa

```
In[16]: nombre=input('¿Cuál es tu nombre?')
¿Cuál es tu nombre?
```

Python espera a que el usuario escriba un arreglo que guardará en la variable `nombre`. Escribe tu nombre, por ejemplo

```
In[17]: nombre=input('¿Cuál es tu nombre?')
¿Cuál es tu nombre? Juan
In[18]:
```

Posteriormente puedes llamar a la variable `nombre`

```
In[18]: nombre=input('¿Cuál es tu nombre? ')
¿Cuál es tu nombre? Juan
In[19]: nombre
Out[19]: 'Juan'
In[20]: print(nombre)
Juan
```


Por default, Python guarda las entradas que recibe del comando input como un arreglo. Es posible verificar esto de la siguiente manera

```
In[22]: type(nombre)
Out[22]: str
```

Aunque, como verás más adelante, es posible cambiar el arreglo a una variable numérica *entera*, o de tipo *float*.

Ingresa las siguientes instrucciones en el prompt de Anaconda para sumar los números 9 y 2.

```
In[23]: n1=input('Ingresa el primer número, de dos, para sumarlos ')
Ingresa el primer número, de dos, para sumarlos 9
In[24]: n2=input('Ingresa el segundo número ')
Ingresa el segundo número 2
In[24]: n1+n2
'92'
```

El valor esperado en la suma era 11. Sin embargo, Python añade al arreglo **9**, el arreglo **2**, produciendo el arreglo **92**. Recuerda que esto se debe a que las variables no son interpretadas como números, Python las interpreta como arreglos y o único que realiza es pegar las dos cadenas de caracteres. A esta operación se le llama *concatenación.*

Para que Python pueda leer los datos como variables numéricas de tipo entero, podemos hacer los siguientes cambios a las instrucciones anteriores

```
In[25]: n1=int(input('Ingresa el primer número, de dos, para sumarlos '))
Ingresa el primer número, de dos, para sumarlos 9
In[26]: n2=int(input('Ingresa el segundo número '))
Ingresa el segundo número 2
In[27]: n1+n2
Out[27]: 11
```

También se pueden cambiar el formato de *string* obtenido por `input`, al formato numérico de tipo *float*, cambiando el comando `int` por `float` en las instrucciones anteriores. Recuerda que el formato *float* permite obtener cantidades que contienen una parte decimal.

2.4 Primeros programas en Python

Para introducir los operadores de comparación, utilizarás dos ejemplos sencillos, pero que están diseñados para que aprendas la estructura de un programa en Python y de la forma en que se emplean varias de las funciones que has utilizado anteriormente. En ambos casos se utiliza la declaración condicional `If`, para remarcar la sintaxis que tiene esta sentencia y forma en que la ejecuta Python.

Ejemplo 2.1 Declaraciones condicionales con If

Esta es una versión sencilla de la declaración condicional `If`. Esta función usa una condición para decidir cuándo ejecutar alguna declaración. En el programa aparecen 6 funciones `If`, cada vez que se ejecutas el programa, tres de las seis condiciones son verdaderas, para que observes el resultado en diferentes condiciones ejecutarás el programa 3 veces:

- Primera corrida: El primer valor es más grande que el segundo
- Segunda corrida: El primer valor es más pequeño que el segundo
- Tercera corrida: Los dos valores iguales

Este ejemplo está incluido en la carpeta del capítulo dos.

El código que sigue el anterior planteamiento es el siguiente.

```
# ejemplo1_c2.py
"""Comparación de enteros mediante sentencias If."""

print('Ingresa tu edad y la de tu mejor amigo,\
para decirte la relación qué hay entre ellas ')
#Lee la primera edad
edad1=int(input('Ingresa tu edad: '))

#Lee la segunda edad
edad2=int(input('Ingresa la edad de tu mejor amigo: '))

if edad1==edad2:
   print(edad1,'es igual',edad2,'\n Tu amigo y tú, \
tienen la misma edad.')

if edad1 !=edad2:
  print(edad1,' es diferente de',edad2,'\n Tu amigo y tú \
tienen edades diferentes.')

if edad1 < edad2:
   print(edad1,' es menor que',edad2,'\n Tu amigo
es mayor que tú.')

if edad1 > edad2:
   print(edad1,' es mayor que',edad2,'\n Tu amigo \
es menor que tú.')

if edad1 <= edad2:
   print(edad1,' es menor o igual a',edad2,'\n Tu amigo \
tiene al menos, tu edad.')

if edad1 > edad2:
       print(edad1,' es mayor que',edad2,'\n Tu amigo \
tiene a lo más, tu edad.')
```


Los ejemplos se encuentran en la carpeta CursoPy que debes tener ya instalada en tu computadora ingresa a la carpeta Cap2 y ejecuta el programa: `ejemplo1_c2.py.`

Para hacerlo desde Anaconda Prompt

Paso 1. Entra al prompt, aparecerá

```
(base) C:\Users\Equipo>
```

Paso 2. Como antes accede a ipython

```
(base) C:\Users\Equipo>ipython
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64
bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.19.0 -- An enhanced Interactive Python. Type '?'
for help.

In [1]:
```

Paso 3. Para ejecutar los programas del capítulo 2 debes estar en la carpeta Cap2. Para acceder a ella teclea `cd documents\Cursopy\Cap2` y presiona Enter.

```
In [1]: cd documents\Cursopy\Cap2
C:\Users\Equipo\documents\Cursopy\Cap2
In [2]:
```

Paso 4. Para ejecutar el primer programa escribe en el snippet `run ejemplo1_c2.py`

Listo. En el prompt aparecerán las solicitudes para ingresar los datos

Algunos resultados son los siguientes

```
Ingresa tu edad y la de tu mejor amigo para decirte la re-
lación qué hay entre ellas
Ingresa tu edad: 25
Ingresa la edad de tu mejor amigo:27
25 es diferente de 27
Tu amigo y tú tienen edades diferentes.
25 es menor que 27
Tu amigo es mayor que tú.
25 es menor o igual a 27
Tu amigo tiene al menos, tu edad
```

```
Ingresa tu edad y la de tu mejor amigo para decirte la re-
lación qué hay entre ellas
Ingresa tu edad: 26
Ingresa la edad de tu mejor amigo:26
26 es igual a 26
Tu amigo y tú tienen la misma edad.
26 es menor o igual a 26
Tu amigo tiene al menos, tu edad
26 es mayor o igual a 26
Tu amigo tiene a lo más, tu edad
```

```
Ingresa tu edad y la de tu mejor amigo para decirte la re-
lación qué hay entre ellas
Ingresa tu edad: 28
Ingresa la edad de tu mejor amigo:22
28 es diferente de 22
Tu amigo y tú tienen edades diferentes.
28 es mayor que 22
Tu amigo es menor que tú.
28 es mayor o igual a 22
Tu amigo tiene a lo más, tu edad
```

Notas

A. Python tiene algunos operadores que se utilizan para comparar valores numéricos. Estas comparaciones pueden ser calificadas como falsas o verdaderas (true, false). True y false son palabras reservadas que emplea Python.

Ejemplos de comparación

```
In[1]: 15<3
Out[1]: False
In[2]: 6>1
Out[2]: True
```

B. La siguiente tabla muestra una lista los diferentes operadores y la prioridad en que se aplican, siendo los primeros en la lista, los de mayor prioridad.

	Operador algebraico	Operador en Python	Descripción	Ejemplo
>	$a > b$	`a > b`	a es mayor que b	`In[3]: 3>6` `Out[3]: False` `In[4]: 3>-6` `Out[4]: True`
<	$a < b$	`a < b`	a es menor que b	`In[5]: 3<6` `Out[5]: True` `In[6]: 3<-6` `Out[6]: False`
≥	$a \geq b$	`a >= b`	e es mayor o igual que b	`In[5]: 3>=3` `Out[5]: True` `In[6]: 3>=6` `Out[6]: False`
≤	$a \leq b$	`a <= b`	a es menor o igual que b	`In[5]: 3<=3` `Out[5]: True` `In[6]: 3<=-6` `Out[6]: False`
=	$a = b$	`a == b`	a es igual que b	`In[5]: 3==6` `Out[5]: True` `In[6]: 3==6` `Out[6]: False`
≠	$a \neq b$	`a != b`	a es diferente que b	`In[5]: 3!=3` `Out[5]: False` `In[6]: 3!=-6` `Out[6]: True`

C. En el programa aparece sí diversas sentencias If. Cuando la sentencia If es calificada por Python como verdadera, realiza una tarea (en el programa anterior, escribe una declaración). Observa lo que ocurre en el siguiente ejemplo

```
In[3]: clima=’frio’
In[4]: if clima==’frio’:
           print(‘Necesito un abrigo’)
           print(‘y beber chocolate’)
Necesito un abrigo
y beber chocolate
```

Para que se puedan realizar las tareas designadas, es importante que las declaraciones conserven la identación o sangría. De otra forma realizará otras tareas o marcará un error.

D. Cuando aplicas un operador de comparación es importante no dejar espacios al momento escribir el código ya que Python lo define como un error

```
In[5]: 10> =-3
  File "<ipython-input-7-6cda1b356d4b>", line 1
    10> =9
        ^
SyntaxError: invalid syntax
```

Existe una diferencia importante entre los operadores '=' y '=='. El primero se utiliza para realizar asignaciones, el segundo es para verificar si dos cantidades son iguales.

Ejemplo 2.2 Valor Máximo de un conjunto de datos

El siguiente es un programa sencillo que utiliza la sentencia If y que calcula el máximo de un conjunto de datos. Observa que los datos de ingreso son transformados al formato float. Revisa la sencillez de la estructura del programa y la forma en que se disponen las condiciones con If, para que Python determine el valor máximo.

```
1	# ejemplo2_c2.py
2	"""Determina el valor máximo de cuatro números"""
3
4	no1=float(input('Ingresa el primer número: '))
5	no2=float(input('Ingresa el segundo número: '))
6	no3=float(input('Ingresa el tercer número: '))
7	no4=float(input('Ingresa el cuarto número: '))
8
9	maximo=no1
10	if no2>maximo:
9	   maximo=no2
10
11	if no3>máximo:
12	   maximo=n3
13
14	if no4>máximo:
15	   maximo=n4
16
17	print('El valor máximo es', maximo)
```

Algunos resultados son los siguientes

```
Ingresa el primer número 15
Ingresa el segundo número 1
Ingresa el tercer número 5
Ingresa el cuarto número -10
El valor máximo es 15.0
```

```
Ingresa el primer número -3.5
Ingresa el segundo número 159
Ingresa el tercer número 0
Ingresa el cuarto número 3.1416
El valor máximo es 159.0
```

```
Ingresa el primer número -0.1
Ingresa el segundo número -3.15
Ingresa el tercer número -18
Ingresa el cuarto número -100
El valor máximo es -0.1
```


El paso inicial es asignar al primer número el valor máximo, este valor lo compara con los otros tres. Si alguna sentencia se cumple, por ejemplo, si el tercer número (`no3`) es mayor que los dos primeros números (sentencia verdadera, línea 13), Python designará a este número como el máximo. Luego lo comparará con el cuarto número (línea 16). Si la sentencia es falsa, el tercer número será el máximo, en otro caso, asignará al cuarto número la variable **`maximo`.**

En los capítulos siguientes seguirás revisando algunos conceptos de programación orientados a la descripción de datos, continua con esta increíble aventura.

2.5 Ejercicios.

1. ¿Cuál es el resultado de las siguientes operaciones en Python?

   a. `In[1]: 5 + 3 ** 2 * 2`
   b. `In[2]: (30 - 1) // (15 - 8)`
   c. `In[3]: 5*(3 - 1) % (7 -4)`

2. En algunas operaciones aritméticas aparecen los llamados paréntesis redundantes. Son llamados así, porque al omitirlos, se observa que no alteran el resultado final. Esto es, los paréntesis se añaden a las operaciones, pero no son necesarios.

   Observa las siguientes operaciones, ¿cuál operación u operaciones tiene paréntesis redundantes?

   a. `Fx= 3 * (x ** 2) + 7`
   b. `Fx= (3 * x ** 2) + 7`
   c. `Fx= 3 * (x ** 2 + 7)`

3. Para escribir la expresión algebraica y=7x-3x^2 en Pyhton. ¿Cuál de las siguientes expresiones no es correcta

   a. `y = 7 * x - 3 * x * x`
   b. `y = (7 * x - 3 ) * x ** 2`
   c. `y = (7 * x) -  ( 3 * x ** 2)`
   d. `y = 7 * x - 3 * x ** 2`

4. Elabora los snippets necesarios para que, en el prompt de Anaconda, aparezca una solicitud del año de nacimiento para el usuario. Con este dato, debes calcular la edad del usuario e imprimir en la pantalla, por ejemplo, algo así

   Usted tiene 29 años.

5. Escribe en el prompt de Anaconda la siguiente instrucción

```
In[4]: print(-2>-100, 5<=5, -7>=-1, -5<0, 7==4)
```

¿Qué es lo que muestra la pantalla?, ¿qué significan estos resultados?

6. ¿A qué tipo de expresiones corresponden los siguientes términos?

    a. 8.0
    b. ‘6.15’
    c. 0.001
    d. 6.15
    e. 31416

7. El siguiente programa solicita al usuario un número entero, y verifica si es un múltiplo de tres.

```
num3=int(input(‘Ingresa un número entero’))
if num3 % 3 == 0
print(‘El número’,num3,’es un múltiplo de 3’)
if num3 % 3 !=0
print(‘El número’,num3,’no es un múltiplo de 3’)
```

Realiza los cambios necesarios al programa para determinar si algún número ingresado por el usuario es múltiplo de siete.

8. Un programa, solicita al usuario ingresar la base y la altura de un rectángulo para calcular el área. Al inicio, el programa tiene las siguientes tres líneas

```
base=input(‘Ingresa la base del rectángulo’)
altura=input(‘Ingresa la altura del rectángulo’)
área_rect=base*altura
```

¿Cuál es el error en el código?

CAPÍTULO 3

Control y desarrollo de programas con Python

3.1 Introducción

A lo largo de este capítulo aprenderás algunas de las estrategias generales que utilizan los expertos de programación para elaborar códigos que resuelven problemas altamente complejos.

Cruzaremos ese camino rodeado de ciclos, lazos y operadores booleanos, así que

"¡En sus marcas!, ¡listo!... PROGRAMA"

3.2 Detrás del telón de la programación, Algoritmos, Pseudocódigo y otras monadas

A pesar de que programar puede ser una tarea fácil, hay programas llenos de complejidad. Por ejemplo, las aplicaciones que utilizas en tu celular son programas realizados por expertos, que, como tú, tuvieron en algún punto geográfico de su historia, la oportunidad de aprender a programar. Pero programar no solo consiste en elaborar códigos. Hay que partir de una idea, planear una estrategia, trazar una ruta; tal vez, equivocarse, replantear la ruta, retomar el camino y llegar a la meta.

En el caso de la programación, trazar la ruta es el equivalente de escribir el código. Por lo que hay algunos pasos previos que no son visibles al momento de utilizar alguna aplicación en el celular.

Iniciaremos este capítulo con un recorrido breve por estas estrategias de planeación.

3.3 Algoritmos

Los libros de programación definen a un algoritmo como una secuencia ordenada de pasos, que han de ejecutarse uno después de otro. Si bien esta definición no está mal del todo, existen otras definiciones más actuales, por ejemplo, Deitel 1 señala que un algoritmo es un procedimiento realizado para resolver un problema, en términos de

1. Las acciones a ejecutar, y
2. El orden en el cual se realizan estas acciones

Piensa en la forma en que preparas un café instantáneo

1. Calientas agua
2. Agregas café
3. Lo bebes

Pero podemos hacer muchas preguntas con respecto a este procedimiento. El agua se puede servir, ¿de una llave?, ¿de un envase?, ¿del mar?, ¿de lluvia? Será calentada: ¿en un vaso?, ¿en una olla?, ¿en un plato?, ¿en las manos? Para calentar el agua se debe hacer: ¿con el sol?, ¿con leña?, ¿con un cerillo?, ¿con una frazada?, etc.

Algunas preguntas pueden parecer exageradas. Incluso fuera de lugar, pero cuando trates de dar instrucciones a la computadora, estas instrucciones deben ser claras y con un orden específico. No puedes beber el agua caliente y agregar café después. Este ejemplo lejos de estar alejado de la realidad, representa uno de los errores más comunes en la programación.

Supongamos que se requiere de un programa que verifique si las personas pueden votar en las siguientes elecciones, por lo que debe verificar si son mayores de 18 años.

Un algoritmo sencillo para hacer esto es:

Paso 1. Ingresar el año de nacimiento
Paso 2. Leer el año
Paso 3. Calcular 2021- el año de nacimiento
Paso 4. Verificar si la resta es mayor o igual a 18
Paso 5. Dictaminar si puede o no puede votar

3.4 Pseudocódigo

El pseudocódigo es el empleo de un lenguaje cotidiano para plantear la estrategia que debe emplearse para resolver un problema. Como el nombre lo dice, es un código falso, pero representa el puente perfecto, para llegar al punto de iniciar a programar

```
Un pseudocódigo del problema anterior puede ser:
En el prompt el usuario ingresa su año de nacimiento
La cantidad debe leerse como un entero
Calcula los años del usuario, años= 2021 – año de nacimiento
Si la variable años es mayor o igual a 25,
el programa debe mostrar que es mayor de edad,
si no se cumple la sentencia debe mostrar que es menor de edad.
```

3.5 Sentencias de Control

Cuando las sentencias en un programa se ejecutan en el orden en que se escriben, se dice que el programa realiza una ejecución secuencial. Existen algunas sentencias en Python que especifican que tarea se debe ejecutar, ya que no específicamente debe ser la siguiente que está escrita. A esto se le llama transferencia de control y están establecidas por las sentencias de control de Python.

Para representar gráficamente lo que hace una sentencia de control se utilizan diagramas de flujo. Algunos de los elementos de estos diagramas que verás en esta sección son los siguientes.



3.6 Sentencias de decisión y repetición

Para tomar decisiones, Python tiene tres tipos de sentencias que ejecutan el código de acuerdo a una condición que puede ser evaluada como falsa o verdadera.

- Sentencia `if`
  Esta sentencia realiza una acción, cuando la condición es verdadera; y la ignora, cuando es falsa.

- Sentencia `if…else`
  Esta sentencia realiza una acción, cuando la condición es verdadera; y realiza la otra acción cuando es falsa.

- Sentencia `if…elif…else`
  Esta sentencia realiza una de diferentes acciones dependiendo como hayan sido calificadas las otras condiciones.

Para aquellos problemas que requieren algún proceso iterativo, Python con dos sentencias de repetición.

- Sentencia `while`
  Esta sentencia indica que la acción se repite mientras que la condición se mantenga como verdadera.
- Sentencia `for`
  Esta sentencia permite repetirla acción para cada elemento en un conjunto de elementos. Útil para realiza una acción, cuando la condición es verdadera; y realiza la otra acción cuando es falsa.

3.7 Sentencia if

Supongamos que una tienda de autoservicio tiene el interés de saber si algún consumidor en particular es mayor de 25 años (población en posibilidad de pagar ciertos servicios), para ofrecerle un seguro de gastos médicos.

Un pseudocódigo del problema anterior puede ser:

```
El usuario proporciona su año de nacimiento
La cantidad debe leerse como un entero
Si la variable 2021 – año de nacimiento, es mayor o igual a 25,
el programa debe mostrar: Posible cliente
```

Observa que, tanto en el pseudocódigo anterior, como en este, después de escribir la condición

Si la variable años es mayor o igual a 25

En la siguiente línea aparece una identación (sangría) para indicar la acción a realizar:

el programa debe mostrar: Posible cliente

Esto se hace con el propósito de que recuerdes que, en el código, al utilizar las sentencias de control de Python es necesario que las acciones a realizar tengan esta identación.

Si algún cliente indica que su año de nacimiento es de 1993 años, el código en Python para el pseudocódigo anterior es

```
In[1]: nac=1993
In[2]: if 2021-nac >= 25:
  ...:print(‘Posible cliente’)
  ...:
Posible cliente
```

En este caso, la persona que nace en 1993, tendrá una edad, en años, de 2021-1993=28. Por lo que la condición 2021-nac se valúa como verdadera y se muestra en pantalla Posible cliente.

Si no se añade la identación.

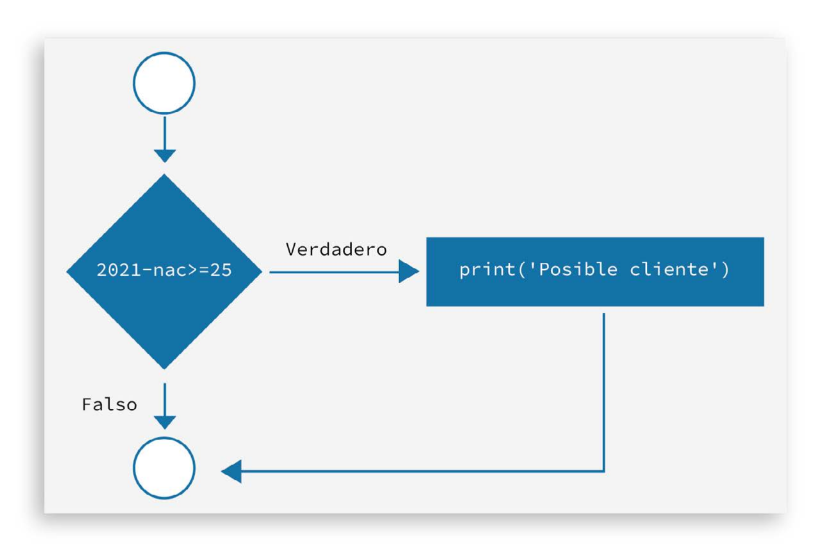
```
In[1]: nac=1993
In[2]: if 2021-nac >= 25:
  ...: print(‘Posible cliente’)
  File "<ipython-input-2-57cdab0cbb89>", line 2
    print('Posible cliente')
    ^
IndentationError: expected an indented block
```

Ocurre un error de identación. Este error también puede ocurrir si tienes más de una acción para realizarse con la condición, pero no tiene la identación.

```
In[1]: nac=1993
In[2]: if 2021-nac >= 25:
   ...:  print(‘Posible cliente’)
   ...:  print(‘Proporcione información del seguro’)
   File "<tokenize>", line 3
     print((‘Proporcione información del seguro’)
     ^
IndentationError: unindent does not match any outer inden-
tation level
```

Trata de aplicar las condiciones de identación. Un programa que no está uniformemente identado resulta difícil de leer.

Para el snippet [2], el diagrama de flujo tiene las siguientes características



Observa que las líneas siguen la dirección del flujo cuando la sentencia es evaluada como verdadera o como falsa. Si es verdadera, la acción se ejecuta. Si es falsa, el programa continua.

3.8 Sentencias if…else, if…elif…else

La sentencia `if…else` realiza dos diferentes tipos acciones, y el resultado depende si la condición es evaluada como verdadera o como falsa.

Un pseudocódigo al problema de buscar clientes con una edad identificada por el formato 2021-años de nacimiento, puede ser:

```
El usuario proporciona su año de nacimiento
La cantidad debe leerse como un entero
```

```
Si la variable 2021 – año de nacimiento, es mayor o igual a 25,
el programa debe mostrar: Posible cliente
en otro caso,
debe mostrar: No es candidato
```

Observa de independientemente, si la acción es evaluada como verdadera o falsa, las acciones se escriben con la identación. A continuación, se muestran dos casos con dos fechas de nacimiento diferentes.

```
In[3]: nac=1985
In[4]: if 2021-nac >= 25:
  ...:  print('Posible cliente')
  ...: else:
  ...:  print('No es candidato')
  ...:
Posible cliente
```

```
In[5]: nac=1999
In[6]: if 2021-nac >= 25:
  ...:  print('Posible cliente')
  ...: else:
  ...:  print('No es candidato')
  ...:
No es candidato
```

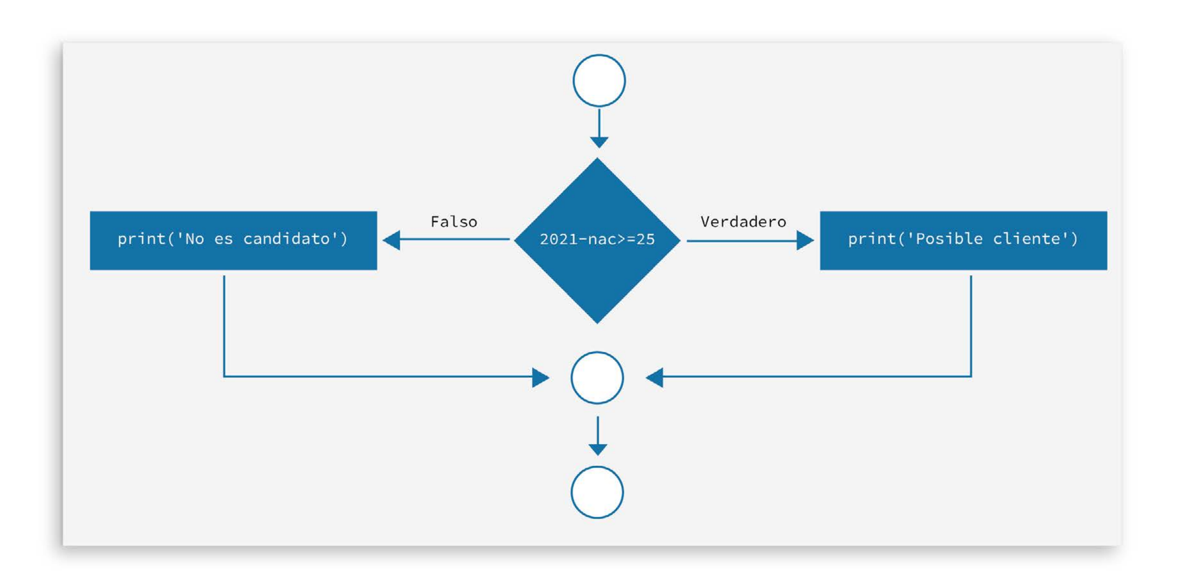
Nota

Para ejecutar el programa no es necesario volver a escribir todo el código. Puedes utilizar las flechas en el teclado (arriba, abajo). En el prompt aparecen los snippets que ya has utilizado.

1. En el snippet [5] mueve las flechas y selecciona, por ejemplo: nac=1999, que ya habíamos utilizado, pero antes de presionar enter, cambia el valor, digamos 1993.
2. Ahora, en el snippet [6], vuelve a utilizar las flechas para seleccionar lo escrito en [4]. Y listo, automáticamente se escribe el código en el prompt y se ejecuta al presionar enter.

```
In[7]: nac=1993
In[8]: if 2021-nac >= 25:
  ...:  print('Posible cliente')
  ...: else:
  ...:  print('No es candidato')
  ...:
Posible cliente
```

El diagrama de flujo para la sentencia if…else, se muestra a continuación.



Cuando la sentencia es evaluada como verdadera la acción `print('Posible cliente')` se ejecuta. En otro caso (si es falsa), el programa ejecuta `print('No es candidato').`

Una forma alternativa de ejecutar las instrucciones anteriores, es asignar las acciones a una variable.

```
In[9]:   nac=1999
In[10]:  edad=2021-nac
In[11]:  if edad >= 25:
   ...:  cliente='Si'
   ...:  else:
   ...:  cliente='No'
   ...:
In[12]:  cliente
Out[12]: 'Si'
```

En este caso, se asignó a la variable edad, la resta **2021-nac.** Y para la sentencia **if… else**, cuando es verdadera se asigna a la variable cliente, el arreglo **'Si'**. Y cuando la sentencia es falsa, se asigna a cliente, el arreglo **'No'.** Al final, se hace la solicitud del resultado en el snippet[10].

Cada condición de la sentencia puede realizar más de una acción.

```
In[13]: nac=2002
In[14]: if 2021-nac >= 25:
   ...:    print('Posible cliente')
   ...:    print('Proporciona información')
   ...: else:
   ...:    print('No es candidato')
   ...:    print('Busca a otra persona')
   ...:
No es candidato
Busca a otra persona
```

Pero si alguna acción queda sin identación, ésta siempre se ejecutará

```
In[13]: nac=2002
In[14]: if 2021-nac >= 25:
   ...:     print('Posible cliente')
   ...:     print('Proporciona información')
   ...: else:
   ...:     print('No es candidato')
   ...:     print('Busca a otra persona')
   ...:
No es candidato
Proporciona información
Busca a otra persona
```

La sentencia `if…elif…else` se emplea para realizar diversas acciones.

Un pseudocódigo clasificar niños, adolescentes, jóvenes, adultos y adultos mayores es:

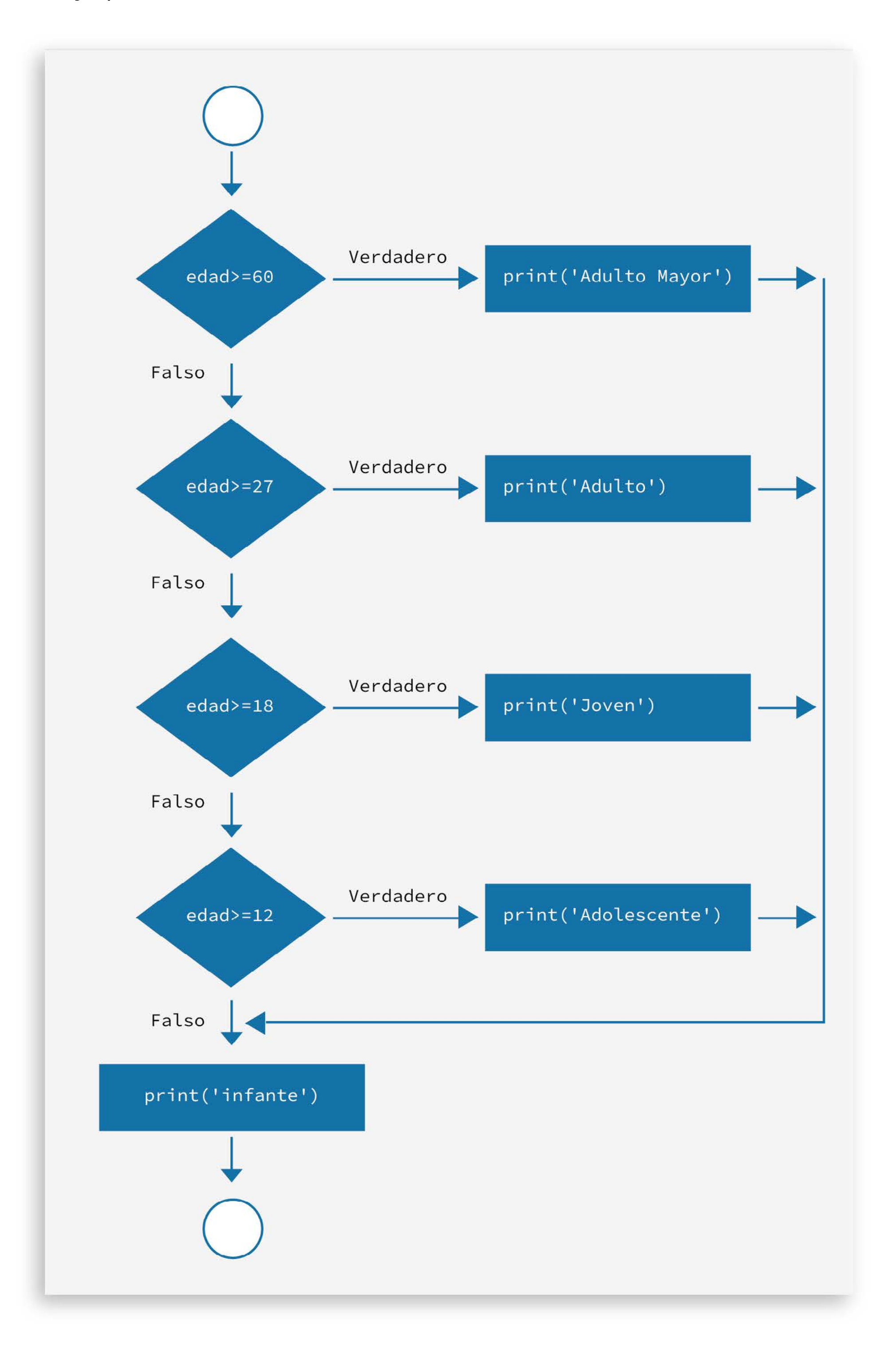
```
Si la edad de la persona es mayor o igual a 60
 Mostrar: Adulto Mayor
Si la edad de la persona es mayor o igual a 27
 Mostrar: Adulto
Si la edad de la persona es mayor o igual a 18
 Mostrar: Joven
Si la edad de la persona es mayor o igual a 12
 Mostrar: Adolescente
En otro caso,
 Mostrar: Infante
```

De acuerdo a este pseudocódigo, solo una acción se ejecutará. El código para clasificar a una persona de 24 años, es el siguiente.

```
In[17]: edad=24
In[18]: if edad >= 60:
   ...:     print('Adulto mayor')
   ...: elif edad >= 27:
   ...:     print('Adulto')
   ...: elif edad >= 18:
   ...:     print('Joven')
   ...: elif edad >= 12:
   ...:     print('Adolescente')
   ...: else:
   ...: print('Infante')
   ...:
Joven
```


Las primeras dos condiciones son falsas, cuando llega a la tercera, Python verifica que `edad >=18` la cual es verdadera. Por lo tanto, muestra **Joven**. Observa que el programa ya encontró la condición verdadera, por lo que ignora las líneas que siguen y ejecuta la acción indicada.

El diagrama de flujo para la sentencia `if…elif…else`, se muestra a continuación.



3.9 Sentencia While

La sentencia **while** permite realiza una acción mientras que una condición se mantenga como verdadera.

Para mostrar la forma en que trabaja esta sentencia, resolveremos el siguiente problema, ¿cuál es la potencia de 4 más grande que 817?

Una propuesta de pseudocódigo es

```
Iniciar con potencia=4
Mientras que potencia sea menor o igual a 817
 Potencia=potencia*4
```

Esto indica que el programa debe seguir calculando potencias de cuatro (sentencia verdadera), hasta que la potencia deje de ser menor o igual a 817 (sentencia falsa)

```
In[1]: potencia=4
In[2]: while potencia <= 817:
  ...:  potencia = potencia * 4
  ...:
In[3]: potencia
1024
```

Con esta sentencia se forma un lazo, o bucle, o ciclo (loop) que repite las instrucciones hasta cumplir cierta condición.

En este caso :

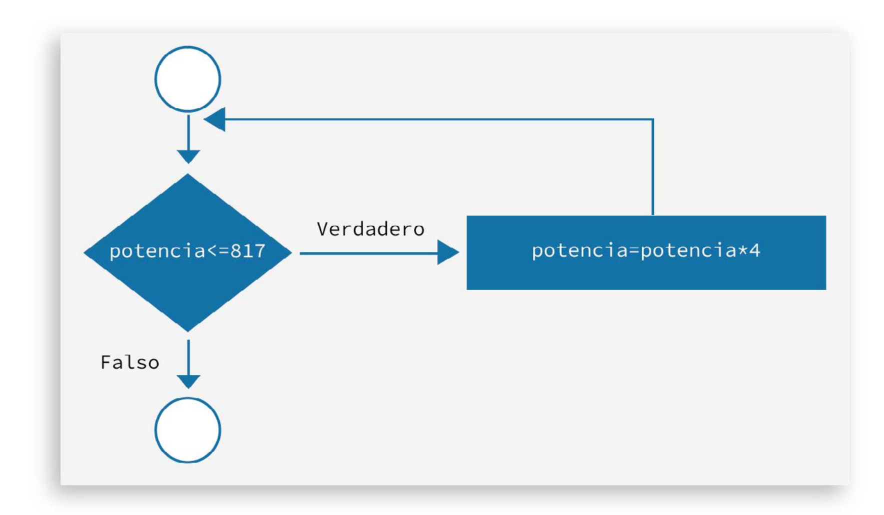
a. El procedimiento inicia con un valor inicial potencia=4, al entrar al lazo, la sentencia evalúa la condición: potencia<=817 (verdadera) y asigna a la nueva variable potencia, el valor de potencia almacenado multiplicado por cuatro, esto es 4^2. Ahora potencia = 4^2=16.

b. Con el valor de potencia =16, la sentencia evalúa la condición: potencia<=817 (verdadera) y asigna a la nueva variable potencia, el valor de potencia almacenado multiplicado por cuatro, esto es 4^3. Ahora potencia = 4^3=64.

c. Este ciclo se repite hasta que la condición potencia<=817 sea falsa.

Un error común en esta sentencia puede ocurrir en la condición. Si la condición no está bien definida puede ocurrir que se cicle la ejecución. Si esto te llega a pasar, puedes detener la ejecución del ciclo con el prompt de Anaconda y teclear Ctrl+c o control+c.

El diagrama de flujo para la sentencia while es:



3.10 Sentencia for

Como en el caso de **while, for** repite una o varias acciones. La sentencia **for** realiza la acción para cada elemento en una sucesión de elementos. Tal sucesión de elementos puede ser una lista de valores, que no necesariamente debe estar ordenada o pueden ser caracteres, ya que una palabra es una lista de caracteres.

Analiza el siguiente ejemplo:

```
In[4]: for caracter in ‘Palabra’:
  ...:     print(caracter, end=’  ‘)
  ...:
P  a  l  a  b  r  a
```

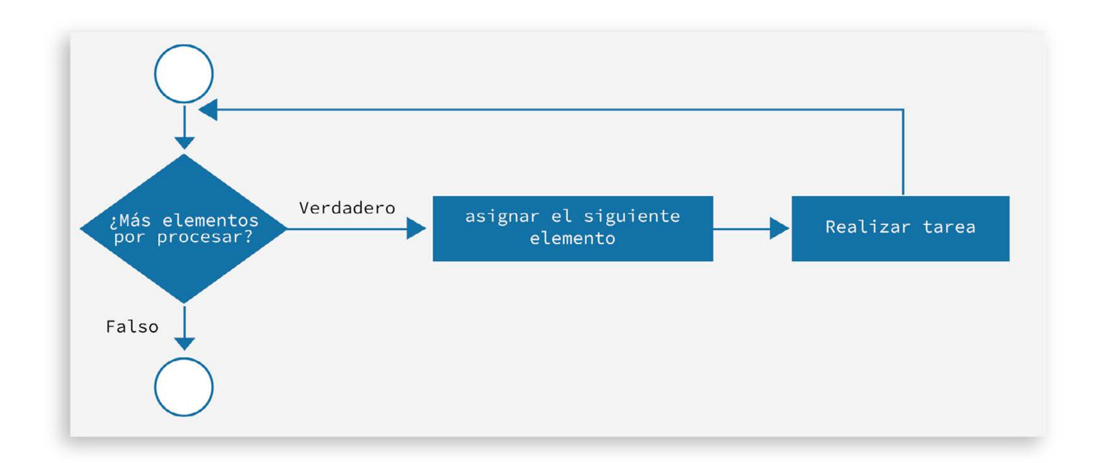
1. En el snippet [1] la sentencia for asigna la letra P del arreglo Palabra y se lo asigna a la variable caracter.
2. Enseguida, muestra el caracter P seguido de dos espacios
3. El siguiente paso es asignar la letra a la variable caracter,
4. Enseguida, muestra el caracter a seguido de dos espacios
5. Esto continua con cada uno de los elementos (caracteres) del arreglo Palabra.

Solo por curiosidad científica, quita el argumento **end=** en el código anterior. ¿Qué observas?

Otra opción para presentar los resultados como en el ejemplo anterior, es utilizar el argumento sep (abreviación de separador). Cuando este argumento es ignorado, por default, Python agrega un caracter de espacio.

```
In[5]: print(25,55,85,115, sep=’, ‘)
25, 55, 85, 115
```

El diagrama de flujo para la sentencia `for` es



Un código sencillo para determinar el valor factorial del número 5 utilizando la sentencia `for`, es el siguiente

```
In[6]: fact=1
In[7]: for nmro in [1,2,3,4,5]:
  ...:  fact=fact*nmro
  ...:
In[8]: fact
Out[8]: 120
```

El resultado final es llamado el valor factorial de 5. En matemáticas, se denota y define como

5!=1*2*3*4*5

Al utilizar instrucciones como `fact=fact*nmro`, en una sentencia for, se dice que los cálculos se encuentran en un proceso *iterativo*, y en cada ocasión que se repite la sentencia, se le llama *iteración*. Estas formas de programar son comunes en aplicaciones orientadas a la estadística y otros métodos numéricos.

Para mejorar estos procesos iterativos, Python tiene incluidos asignadores de aumento para abreviar códigos como

fact=fact*nmro.

Aquí está la muestra

```
In[9]: fact=1
In[10]: for nmro in [1,2,3,4,5]:
  ...:  fact *= nmro
  ...:
In[11]: fact
Out[11]: 120
```


Otros asignadores de aumento se muestran en la siguiente tabla

Operación	Con el asignador
a = a+8	a+=8
c = c*4	c*=4
e = e/5	e/=5
g = g%3	g%=3

Operación	Con el asignador
b = b-1	b-=1
d = d**2	d**=2
f = f//2	f//=2

Cuando la sucesión de elementos en una sentencia `for` es una sucesión ordenada, es posible utilizar la función range, por ejemplo

```
In[12]: for contador in range(5)
  ...:      print(contador, end=' ')
  ...:
0 1 2 3 4
```

Observa que la función `range(5)` genera una lista de valores que inicia en cero y finaliza, un entero previo al **5**.

El siguiente código muestra la forma de utilizar la sentencia for aplicado a una situación común en las tiendas de autoservicio.

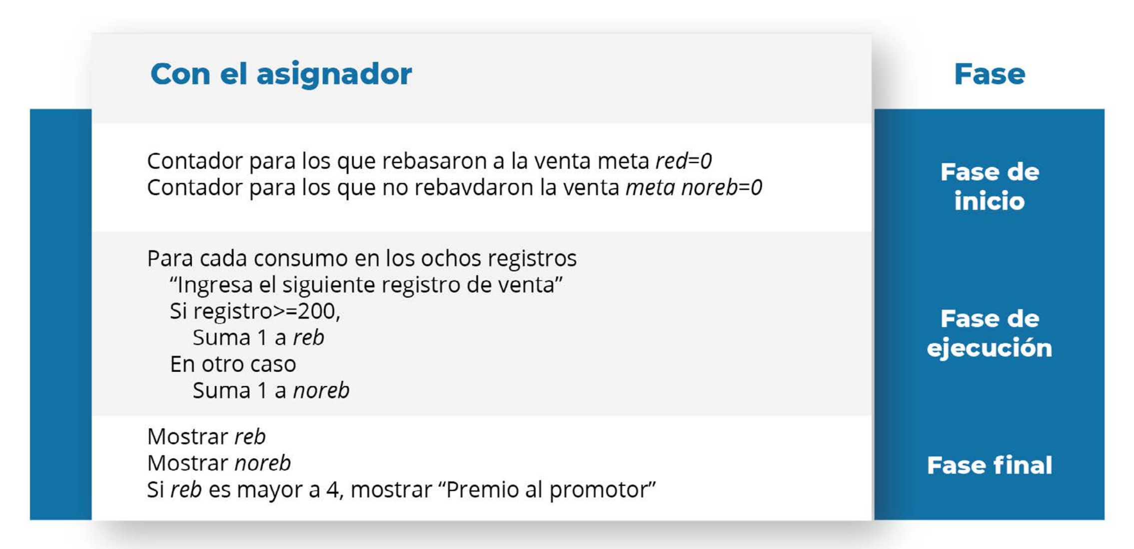
Ejemplo 3.1

A una pequeña tienda de autoservicio entran 8 personas a consumir diversos productos. La siguiente lista describe los consumos (en pesos) realizados por los clientes

[130,85,210,45,153,78.5,264.5,94]

Elabora un programa que calcule el consumo promedio realizada por estos clientes.

Para resolver este ejercicio es necesario



Con el asignador	Fase
Contador para los que rebasaron a la venta meta *red=0* Contador para los que no rebavdaron la venta *meta noreb=0*	Fase de inicio
Para cada consumo en los ochos registros “Ingresa el siguiente registro de venta” Si registro>=200, Suma 1 a *reb* En otro caso Suma 1 a *noreb*	Fase de ejecución
Mostrar *reb* Mostrar *noreb* Si *reb* es mayor a 4, mostrar “Premio al promotor”	Fase final

En la columna de la derecha se anexa la fase en que se realizan las diferentes tareas en el programa. Aunque esta clasificación no afecta en nada el resultado final, es de gran ayuda al momento de planificar las tareas que debe realizar el programa.

El código que sigue el anterior planteamiento es el siguiente.

```
1    # ejemplo1_c3.py
2    """Uso de la sentencia for para calcular el consumo promedio"""
3
4     #Fase inicial
5     suma_tot=0
6     contador_clientes=0
7     consumos=[130,85,210,45,153,78.5,264.5,94]
8
9     #Fase de ejecución
10    for consumo in consumos:
11       suma_tot += consumo
12       contador_clientes += 1
13
14    #Fase final
15    consumo_prom=suma_tot/contador_clientes
16    print(f'El consumo promedio es de {consumo_prom}')
```

```
El consumo promedio es de 132.5
```

Observa que en la línea 16, el programa utiliza el formato de cadena (formatted string o f-string), cuya función es la de cambiar el valor numérico del promedio en una cadena de caracteres. En los programas siguientes adelante utilizaremos otras características de este formato para la salida de la información.

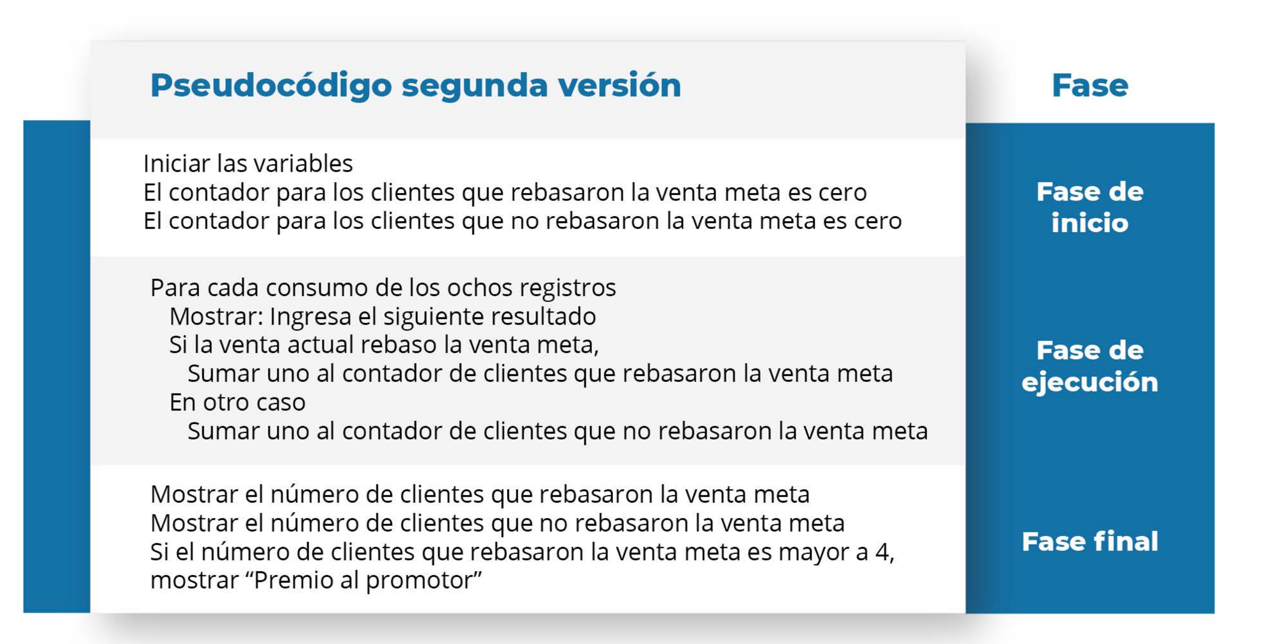
Ejemplo 2

Supongamos que ahora en la tienda solicitan el gasto promedio de los clientes que ingresan en un lapso de 4 horas. Así, ya no es posible dejar fija la lista pues el número de clientes puede ser grande, pequeño, pero incluso, puede ser cero.

Cuando esto ocurre es posible utilizar un valor especial, al que llamaremos valor centinela. Este valor tiene el propósito de indicarle al programa que no habrá más datos de ingreso. Cuando se implementa esta estrategia el programa seguirá solicitando datos hasta que sea introducido el valor centinela.

Un valor centinela no debe tener las mismas características de los datos de ingreso. Para nuestro ejemplo, una posibilidad es seleccionar el valor -1 o cualquier otro valor negativo.

Algunas mejoras al programa `ejemplo1_c3.py` son



Pseudocódigo segunda versión	Fase
Iniciar las variables El contador para los clientes que rebasaron la venta meta es cero El contador para los clientes que no rebasaron la venta meta es cero	Fase de inicio
Para cada consumo de los ochos registros Mostrar: Ingresa el siguiente resultado Si la venta actual rebaso la venta meta, Sumar uno al contador de clientes que rebasaron la venta meta En otro caso Sumar uno al contador de clientes que no rebasaron la venta meta	Fase de ejecución
Mostrar el número de clientes que rebasaron la venta meta Mostrar el número de clientes que no rebasaron la venta meta Si el número de clientes que rebasaron la venta meta es mayor a 4, mostrar “Premio al promotor”	Fase final

El programa con las modificaciones anteriores es

```
# ejemplo2_c3.py
"""Uso de la sentencia for con centinela para el consumo promedio"""

  #Fase inicial
  suma_tot = 0
  contador_clientes = 0

  #Fase de ejecución
   consumo = float(input('Ingresa el consumo, -1 para finalizar: '))
```

```
10
11      while consumo != -1:
12        suma_tot += consumo
13        contador_clientes += 1
14        consumo = float(input('Ingresa el consumo, -1 para
finalizar: '))
15
16      #Fase final
17      if consumo != 0:
18         consumo_prom=suma_tot/contador_clientes
19         print(f'La venta promedio es de {consumo_prom:.2f}')
20      else:
        print('No hubo consumidores')
```

```
Ingresa el consumo, -1 para finalizar: 598
Ingresa el consumo, -1 para finalizar: 365
Ingresa el consumo, -1 para finalizar: 1000
Ingresa el consumo, -1 para finalizar: 100
Ingresa el consumo, -1 para finalizar: 400
Ingresa el consumo, -1 para finalizar: 365
Ingresa el consumo, -1 para finalizar: -1
La venta promedio es de 471.33
```

Observa que en la línea 19

```
print(f'La venta promedio es de {consumo_prom:.2f}')
```

vuelve a aparecer una cadena formateada, pero en este caso el consumo promedio aparece con la indicación `consumo_prom:.2f`, cuya finalidad es que el resultado se presente con dos cifras decimales significativas.

Ejemplo 3

En una tienda de autoservicio hay un promotor en la sección de productos lácteos. La empresa para la que trabaja, le otorga un premio económico si rebasa $200 de ventas (venta meta) por cliente. La empresa cuenta con un registro en donde están marcadas las ventas de 8 clientes que adquirieron algunos de los productos de la marca. Por cada cliente que rebasó la cantidad meta se escribe 1, y si no la rebasa se escribe 2. Si número de clientes que rebasan la venta meta es más de la mitad, se le otorga el premio al promotor. Se requiere de un programa que señale si al promotor se le debe asignar el premio económico.

Antes de escribir el programa, es necesario hacer algunas reflexiones que perfeccionarán el pseudocódigo y facilitarán las tareas y el orden en que deben realizarse.

Pseudocódigo primera versión	Fase
Iniciar las variables	Fase de inicio
Ingresar las ocho ventas, contando las que rebasaron la meta y las que no lo hicieron	Fase de ejecución
Analizar la información para decidir si se otorga, o no, el premio al promotor	Fase final

A este esquema es posible designar más detalles

Pseudocódigo primera versión	Fase
Iniciar las variables El contador para los clientes que rebasaron la venta meta es cero El contador para los clientes que no rebasaron la venta meta es cero	Fase de inicio
Para cada uno de los ocho registros Mostrar: Ingresa el siguiente resultado Si la venta actual rebaso la venta meta, Sumar uno al contador de clientes que rebasaron la venya meta En otro caso Sumar uno al contador de clientes que no rebasaron la venta meta	Fase de ejecución
Mostrar el número de clientes que rebasaron la venta meta Mostrar el número de clientes que no rebasaron la venta meta Si el número de clientes que rebasaron la venta meta es mayor a 4, mostrar "Premio al promotor"	Fase final

Una tercera versión mejorada es

Pseudocódigo segunda versión	Fase
Contador para los que rebasaron la venta meta reb=0 Contador para los que no rebasaron la venta meta noreb=0	Fase de inicio
Para cada consumo en los ocho registros "Ingresa el siguiente registro de venta" Si registro>=200, Suma 1 a reb En otro caso Suma 1 a noreb	Fase de ejecución
Mostrar reb Mostrar noreb Si reb es mayor a 4, mostrar "Premio al promotor"	Fase final

Aunque parecen un tato repetitivos estos últimos tres cuadros tiene el propósito de que observes la forma en que se plantean las tareas en las diferentes etapas, y como en la parte final prácticamente se ha construido el código.

El programa con las últimas indicaciones es:

```
# ejemplo3_c3.py
"""Uso de sentencias anidadas"""

#Fase inicial
reb = 0
noreb = 0

#Fase de ejecución
for registro in range(1,9):
  resultado = int(input('Ingresa el resultado (1=si >=200, 2= si <200) '))
  if resultado==1:
     reb+=1
  else:
     noreb+=1

#Fase final
 print('Número de clientes que rebasaron la venta meta',reb)
 print('Número de clientes que no rebasaron la venta meta',noreb)
if reb > 4:
   print('Premio al promotor')
```

```
Ingresa el resultado (1=si >=200, 2= si <200)1
Ingresa el resultado (1=si >=200, 2= si <200)1
Ingresa el resultado (1=si >=200, 2= si <200)2
Ingresa el resultado (1=si >=200, 2= si <200)2
Ingresa el resultado (1=si >=200, 2= si <200)1
Ingresa el resultado (1=si >=200, 2= si <200)1
Ingresa el resultado (1=si >=200, 2= si <200)2
Ingresa el resultado (1=si >=200, 2= si <200)1
Número de clientes que rebasaron la venta meta 5
Número de clientes que no rebasaron la venta meta 3
Premio al promotor
```

```
Ingresa el resultado (1=si >=200, 2= si <200)2
Ingresa el resultado (1=si >=200, 2= si <200)2
Ingresa el resultado (1=si >=200, 2= si <200)2
Ingresa el resultado (1=si >=200, 2= si <200)2
Ingresa el resultado (1=si >=200, 2= si <200)2
```



```
Ingresa el resultado (1=si >=200, 2= si <200)1
Ingresa el resultado (1=si >=200, 2= si <200)2
Ingresa el resultado (1=si >=200, 2= si <200)1
Número de clientes que rebasaron la venta meta 2
Número de clientes que no rebasaron la venta meta 6
```

Notas

1. Dado que en las condiciones indican un número fijo de personas, el programa utiliza un ciclo for como en el ejemplo 1, en lugar de un ciclo while.

2. En el código del programa 3, aparece la función range. Esta función genera una lista de valores ordenados, hacia adelante o hacia atrás. Observa los siguientes ejemplos

```
In[13]: for x in range(-5,6):
  ...:       print(x, end = ' ')
  ...:
-5 -4 -3 -2 -1 0 1 2 3 4 5
```

Este ciclo genera una lista de valores enteros consecutivos, desde -5 hasta 5. La opción range no incluye el extremo el valor máximo en el rango de -5 a 6.

```
In[14]: for x in range(-5,6,2):
  ...:       print(x, end = ' ')
  ...:
-5 -3 -1 1 3 5
```

Este ciclo genera una lista de valores enteros, desde -5 hasta 5, pero con un tamaño de paso 2. La opción for interpreta perfectamente el recorrido del ciclo con números enteros. Incluso si es hacia atrás.

```
In[15]: for x in range(5,-5,-3):
  ...:       print(x, end = ' ')
  ...:
5 2 -1 -4
```

Algunas ocasiones es necesario expresar los resultados mediante una tabla.

El siguiente ejemplo como crece un capital de $25,000 invertido en una cuenta bancaria que genera un interés simple de 6.5% anual. En este ejemplo se asume que no hay retiros de capital en un periodo 12 años.

La fórmula para calcular el capital C_n después de invertir un monto inicial de C_0, en n periodos, a una tasa de interés de i*100% es

*C_n=C_0 (1+n*i)*

```
In[16]: Co=13500
In[17]: i=0.063
In[18]: n=12
In[19]: for año in range(1,n+1):
  ...:     Capital=Co*(1+año*i)
  ...:     print(f'{año:>2}{Capital:>10.2f}')
1      14350.50
2      15201.00
3      16051.50
4      16902.00
5      17752.50
6      18603.00
7      19453.50
8      20304.00
9      21154.50
10     22005.00
11     22855.50
12     23706.00
```

En la instrucción

```
print(f'{año:>2}{Capital:>10.2f}')
```

esta implementado un f-string (formato de cadena) y tiene dos marcadores de posición que le dan el formato a la salida.

El marcador `{año:>2}` usa el símbolo `>2` para indicar que el año debe estar alineado a la derecha, en un campo de ancho 2. El ancho del campo indica el número de posiciones (medidos en caracteres) que se utilizarán para mostrar el valor.

En el otro marcador `{Capital:>10.2f}` la especificación `10.2f` le da formato a la cantidad, en este caso como un número de punto flotante `(f)`, alineado a la derecha `(>)` y con una parte decimal de dos dígitos `(.2)`.

Python brinda opciones para alterar el flujo de un ciclo, ya sea `for` o `while`. Observa el siguiente ejemplo:

```
In[20]: for n in range(1,11):
  ...:     if n==8:
  ...:        break
  ...:     print(n,end=' ')
  ...:
1 2 3 4 5 6 7
```


Con estas instrucciones Python genera una sucesión de números enteros de 1 a 10, pero el ciclo finaliza cuando el contador `n` del ciclo `for` es igual a 8.

Otra declaración complementaria para `break` es `continue`, que utilizaremos posteriormente.

3.11 Operadores Booleanos

En programación existen operadores booleanos que permiten poner condiciones que mezclan operadores como los que ya has utilizado >, <, >=, <=, == y !=.

Operador Booleano and

Aquí un ejemplo

```
In[1]: registro_email='Si'
In[2]: salario_mens=25000
In[3]: if registro_email=='Si' and salario_mens>=20000:
  ...:     print('Posible cliente. Enviar información')
  ...:
Posible cliente. Enviar información
```

La función clave en este código en **and**. La sentencia `if` será calificada como verdadera, solo cuando `registro_email e ingreso_mens>=20000` sean verdaderas.

Esta regla está definida por la lógica proposicional. La siguiente tabla muestra la forma en que

Proposición 1	Proposición 2	Proposición 1 y proposición 2
V	V	V
V	F	F
F	V	F
F	F	F

trabaja la lógica de este operador.
Esto indica que, si alguna proposición o condición es calificada como falsa, la sentencia if la calificará como falsa, y en consecuencia será verdadera, solo si ambas son verdaderas.

Operador Booleano or

Proposición 1	Proposición 2	Proposición 1 y proposición 2
V	V	V
V	F	V
F	V	V
F	F	F

Este operador se utiliza para verificar si al menos una de dos condiciones (pueden ser más) es verdadera.

```
In[4]: título='Si'
In[5]: experiencia_años=4
In[6]: if título=='Si' or experiencia_años>=5:
  ...:     print('Realizar entrevista')
  ...:
Realizar entrevista
```

El snippet [6] tiene dos condiciones que título=='Si' o que experiencia_años>=5. Como la primera es verdadera, la sentencia if la evalúa como verdadera.

La tabla de verdad para el operador lógico or es la siguiente

Esto indica que, si alguna proposición es calificada como verdadera, la sentencia if la calificará como verdadera, y en consecuencia será falsa, solo si ambas son falsas.

3. 12 Ejercicios

1. Modifica el script del ejemplo 3. Con el objetivo de validar los datos ingresados. Añade un contador que registre los casos en los que no se registró ni 1, ni 2. Al final el programa debe mostrar también el número de registros erróneos.

2. Utiliza un f-string para presentar dos columnas como las siguientes.

```
Raíz      Factorial
1.0000        1.00
1.4142        2.00
1.7321        6.00
2.0000       24.00
2.2361      120.00
```


Donde la primera columna tiene las raíces cuadradas de los primeros cinco números naturales y la segunda los factoriales de las mismas cantidades.

TIP: Si quieres que se muestren los títulos (Raíz y Factorial), puedes escribir la instrucción antes de ejecutar el ciclo en los cálculos

3. Modifica el script anterior, para que ahora los números están alineados de la siguiente forma

```
Raíz      Factorial
1.0000         1.0
1.4142         2.0
1.7321         6.0
2.0000        24.0
2.2361       120.0
```

4. ¿Qué es lo que realiza el siguiente código?

```
for row in range(5):
    for column in range(5):
        print('a' if row %2==0 else 'b', end=' ')
    print()
```

5. Modifica el script anterior para que se muestre el siguiente arreglo

```
0 1 0 1 0
0 1 0 1 0
0 1 0 1 0
0 1 0 1 0
0 1 0 1 0
```

6. En el script del problema, la sentencia if se evalúa al verificar si la fila es par. Modifica esta sentencia para comparar filas y columnas, para obtener arreglos como los siguientes

```
1 1 1 1 1   1
  1 1 1 1   1 1
    1 1 1   1 1 1
      1 1   1 1 1 1
        1   1 1 1 1 1
```

7. ¿Cuánto miden tus amigos? Elabora un programa utilizando la sentencia while, que permita ingresar una cantidad arbitraria de estaturas (todo dependerá de cuantos amigos tengas), para calcular el promedio de la estatura.

8. Elabora un programa que indique cuántos de tus amigos miden lo mismo que tú, pero que también indique cuántos son más altos y cuántos tienen una estatura menor a la tuya.

CAPÍTULO 4

Funciones

4. 1 Introducción

En programación, como en muchas otras áreas, la mejor receta para resolver un problema grande y complejo, es resolver problemas pequeños, que de manera conjunta puedan dar solución a todos los detalles involucrados en este enorme problema.

En este capítulo aprenderás a pulir muchos de esas pequeñas soluciones que en poco tiempo podrán resolver esos grandes retos que rodean al análisis de grades bases de datos.

Así que, bienvenido otra vez a este espectáculo de Python.

Última llamada: tres, dos, uno, ..., programa.

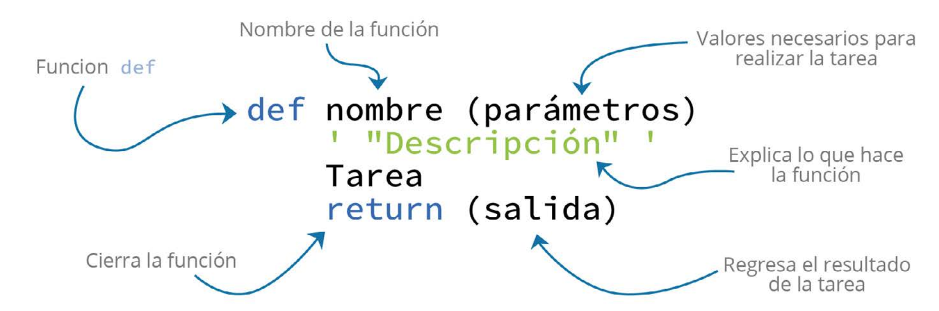
4.2 Función def

Revisa atentamente el siguiente código:

```
[1]:  def raiz_cuad(número):
      """Calcula la raíz cuadrada de un número."""
      return número**(1/2)
[2]:  raiz_cuad(49)
[2]:  7.0
```

Aquí está definida una función llamada raiz_cuad, y la tarea que realiza es calcular la raíz cuadrada de un valor llamado número.

La forma general para definir funciones sigue la siguiente sintaxis



- **`def`,** es una función reservada por Python y tiene como tarea definir nuevas funciones.
- **`nombre`,** es el nombre de la función que quieres definir. Ten la precaución de no utilizar las palabras reservadas por Python.
- parámetros, es una lista de elementos de entrada, necesarios para que la tarea se ejecute.
- En programación siempre es recomendable anexar una descripción sencilla de la tarea que realiza el programa. Siguiendo esta recomendación, observa que en el snippet [1] viene una leyenda con esta descripción.
- Tarea, aquí están descritos todas las sentencias o cálculos que han de realizarse para llegar al resultado esperado. Observa que en el código esta parte está presente en return, ya que la tarea consiste en realizar un solo cálculo.
- return, cuando la función se ejecuta, las tareas terminan con esta instrucción y muestra los resultados obtenidos.

Notas

Al invocar una función, los parámetros que la alimentan son llamados variables locales. Estás variables existen solo durante la ejecución del programa, por lo no se almacenan y se pierden al momento de terminar la tarea.

Al crea una función a través de la función def, Python documenta automáticamente algunas características de la función. Estos detalles se pueden observar.

```
[3]:   raiz_cuad?

Signature: raiz_cuad(número)
Docstring: Calcula la raíz cuadrada de un número.
File:      c:\users\equipo\practicas\<ipython-input-11-14886b56b74d>
Type:      function
```


4.3 Funciones con Múltiples Parámetros

El siguiente código define la raíz cuadrada más grande de tres números

```
[4]:  def raiz_min(num1,num2,num3):
      """Calcula la raíz cuadrada más pequeña de 3 valores"""
      r_num1=num1**(1/2)
      r_num2=num2**(1/2)
      r_num3=num3**(1/2)
      raiz_min=r_num1
      if r_num2<raiz_min:
         raiz_min=r_num2
      if r_num3<raiz_min:
         raiz_min=r_num3
      return raiz_min
[5]:   raiz_min(9,25,4)
[5]:   2.0
[6]:   raiz_min(74,73,75)
[6]:   8.54400374531753
[7]:   raiz_min(7.5,0.3,0)
[7]:   0.0
```

4.4 Números aleatorios

Unos de los temas que dieron origen a la teoría de probabilidad fueron los juegos de azar. Aquí una muestra de puede hacer Python.

El siguiente código muestra los resultados de simular el lanzamiento de una moneda, supongamos que si sale 1 es como si el resultado de un lanzamiento es cara, y si muestra un 0, el resultado es un águila.

```
[1]:   import random

[2]:   for tirada in range(10):
    print(random.randrange(0,2),end=' ')
 1 1 0 0 0 0 1 0 0 1
```

Ejecuta nuevamente el código. Observarás que los resultados no son los mismos. Esto se debe a que la función *random* actualiza un valor especial llamado semilla (seed) para generar números pseudo aleatorios, por lo que cada vez que se ejecuta, el programa muestra resultados diferentes.

Ejemplo 1

Para repetir este experimento aleatorio, que consiste en lanzar una moneda 100,000 veces, no es tan relevante observar una página completa con ceros y unos. Para entender mejor el comportamiento del experimento, el siguiente código (ejemplo1_C4), muestra la frecuencia con la que ocurren las caras (unos) y las águilas (ceros).

Este ejemplo está incluido en la carpeta del capítulo cuatro.

```
# ejemplo2_c4.py
"""100,000 lanzamientos de una moneda."""
import random

#Contadores para las frecuencias

frecuencias_unos=0
frecuencias_ceros=0

for tirada in range(100_000):
   resultado = random.randrange(0,2)
   if resultado == 1:
      frecuencias_unos += 1
   else:
      frecuencias_ceros += 1

print(f'1=Cara,0=Águila{"Frecuencia":>13}')
print(f'{1:>14}{frecuencias_unos:>13}')
print(f'{0:>14}{frecuencias_ceros:>13}')
```

```
1=Cara, 0=Águila   Frecuencia
             1        50088
             0        49912
```

En este ejemplo, como en el anterior, la función `randrange` genera números enteros aleatorios en el rango [0, 1].

Recuerda que `range` excluye el valor 2.

Algunas veces es necesario reproducir los mismos números aleatorios. Esto es posible si en la función random seed se deja fija la semilla.

```
[3]:   random.seed(1)
[4]:   for tirada in range(10):
    print(random.randrange(0,2),end=' ')
 0 0 1 0 1 1 1 1 0 0
```


Al cambiar la semilla

```
[5]:   random.seed(21)
[6]:   for tirada in range(10):
    print(random.randrange(0,2),end=' ')
 0 1 1 1 1 0 1 0 0 0
```

Si pruebas con la semilla del snippet [3]

```
[7]:   random.seed(1)
[8]:   for tirada in range(10):
    print(random.randrange(0,2),end=' ')
 0 0 1 0 1 1 1 1 0 0
```

Se repiten los resultados.

Utilizaremos nuevamente la función random.randrange. Pero ahora para reproducir un juego clásico de casino: "El gran 8"

Ejemplo 2

La Reglas del juego. Debes lanzar un par de dados. Si la suma de las caras es un 8, ganas. Si sale 7, pierdes. Si no ha salido, ni 8, ni 7, puedes seguir lanzando. Si sale 8 ganas, pero si en algún otro lanzamiento sale 7, pierdes.

```
# ejemplo2_c4.py
""" Juego simulado del Gran 8."""

import random

def lanzar_dados():
   dado1 = random.randrange(1,7)
   dado2 = random.randrange(1,7)
   return(dado1,dado2)
#Contadores para las frecuencias

def mostrar_dados(dado):
   dado1,dado2 = dado
   print(f'Resultado de la jugada {dado1}+{dado2}={sum(dado)}')

valores_logrados = lanzar_dados()

mostrar_dados(valores_logrados)

suma_dados = sum(valores_logrados)
```

```
21
22      if suma_dados in (8):
23         status_jugador = 'Ganaste'
24      elif suma_dados == 7:
25         status_jugador = 'Perdiste'
26      else:
27         status_jugador = 'Continuar'
28         #puntos_logrados=suma_dados
29         print('Sigue probando, encuentra al Gran 8')
30
31      while status_jugador == 'Continuar':
32         valores_logrados = lanzar_dados()
33         mostrar_dados(valores_logrados)
34         suma_dados = sum(valores_logrados)
35         if suma_dados in (6,8):
36            status_jugador = 'Ganaste'
37         elif suma_dados == 7:
38            status_jugador = 'Perdiste'
39
40      if status_jugador == 'Ganaste':
41         print('Ganaste')
42      else:
43         print('Perdiste')
```

```
Resultado de la jugada 6+5=11
Sigue probando, encuentra al Gran 8
Resultado de la jugada 6+2=8
Ganaste
```

```
Resultado de la jugada 1+1=2
Sigue probando, encuentra al Gran 8
Resultado de la jugada 6+4=10
Resultado de la jugada 6+5=11
Resultado de la jugada 2+1=3
Resultado de la jugada 2+3=5
Resultado de la jugada 1+3=4
Resultado de la jugada 3+4=7
Perdiste
```

```
Resultado de la jugada 2+4=6
Sigue probando, encuentra al Gran 8
Resultado de la jugada 3+1=4
Resultado de la jugada 4+3=7
Perdiste
```

Algunas observaciones importantes con respecto al código del ejemplo anterior.

En la línea 6, la función `def lanzar_dados():` es llamada varias veces, cuando aparece el paréntesis vacío, indica que la función no necesita parámetros para ejecutarse. Este tipo de funciones pueden regresar uno o varios valores. En este caso `lanzar_dados` regresa dos valores y los coloca en tupla (en este caso, un par ordenado) que contiene los valores de cada lanzamiento.

La función `mostrar_dados` en la línea 18, separa los valores en el arreglo. Estos valores se asignan, como tupla, a una variable `(dado)`, al separarlas por una coma.

En la línea 14 la función `print`, muestra un *f-string* que contiene los valores individuales y su suma. Para realizar la suma en la tupla dado se utiliza la función `sum`, cuya tarea es sumar los elementos en un arreglo numérico.

Un detalle importante, es que las funciones `lanzar_dados` y `mostrar_dados` contienen variables locales `dado1, dado2,` y no se colapsan entre ellas, ya que son funciones diferentes que se ejecutan solo en el bloque donde están definidas.

En las líneas 6 a la 14 se realiza la fase inicial, aquí se determinan de manera aleatoria los valores simulados que pueden ocurrir al lanzar dos dados.

Las líneas 18 y 20 hacen la solicitud de mostrar el resultado del primer lanzamiento del par de dados.

A continuación, inicia la fase de ejecución. Aquí se establece el estatus del jugador, si ganó, si perdió o si requiere continuar.

Si el resultado es 8, el programa indica que el jugador ganó. Si el resultado es 7, el jugador perdió. Si no es ninguna de las anteriores opciones el estatus del jugador es continuar, por lo que seguirá jugando.

El último ciclo (línea 31) simula diferentes partidas hasta que el jugador gane o pierda.
La fase final del programa muestra el estatus final.

4.5 Funciones sin parámetros y múltiples parámetros

Al iniciar este capítulo, viste un ejemplo en donde era posible encontrar la raíz cuadrada de un número.

```
[1]:   def raiz_cuad(número):
           """Calcula la raíz cuadrada de un número."""
           return número**(1/2)
[2]:   raiz_cuad(49)
[2]:   7.0
```

Esta forma de definir la función, indica que es necesario ingresar un valor (la variable número) para que la función se ejecute. Cuando dejas el espacio vacío, Python señala que existe un error.

```
[3]:   raiz_cuad()
```

```
---------------------------------------------------------
```

```
TypeError                 Traceback (most recent call last)
<ipython-input-2-403c50eb3258> in <module>
----> 1 raiz_cuad()

TypeError: raiz_cuad() missing 1 required positional argu-
ment: 'número'
```

Para evitar este tipo de errores, es posible asignar valores iniciales que luego pueden ser cambiados. Por ejemplo.

```
[4]:  def raiz_cuad(número=1):
      """Calcula la raíz cuadrada de un número."""
      return número**(1/2)

[5]:   raiz_cuad()

[5]:   1.0

[6]:   raiz_cuad(27)

[6]:   5.196152422706632
```

A estos valores iniciales se les llama valores por default, y aparecen de manera continua en muchas aplicaciones.

Por otro lado, hay funciones que pueden necesitar más de un parámetro.

El siguiente ejemplo calcula el promedio de calificaciones. Las calificaciones pueden ser de diferentes personas o diferentes asignaturas, pero además puede tener una cantidad arbitraria de elementos.

```
[7]:   def promedio_calif(*calif):
    return sum(calif)/len(calif)

[8]:   promedio_calif(9,8,8,9.5,9.7,10,6)

[8]:   8.6

[9]:   promedio_calif(7,7.3,8.1)

[9]:   7.466666666666666
```

En este ejemplo además de utilizar el parámetro `*calif`, emplea la función sum, cuya tarea es sumar los elementos de la tupla `calif`, y `len`, cuya tarea es determinar el número de elementos en la tupla `calif`.

Observa que no hay una cantidad límite para seguir agregando calificaciones.

4.6 Alcance local y global

En los últimos ejemplos has utilizado funciones que trabajan con variables locales. Por ejemplo:

```
[1]:  def raiz_cuad(número = 1):
      """Calcula la raíz cuadrada de un número."""
      return número ** (1/2)

[2]:  raiz_cuad(64)

[2]:  8.0
```

La función `raiz_cuad` aplica una tarea al valor 64, pero, este valor se pierde para cálculos posteriores. De manera que 64 trabaja solo en este bloque, que es justo la forma en que se definen las variables de alcance local.

Las variables de alcance global pueden ser llamadas para trabajar en diferentes bloques. Aquí está un ejemplo de una variable global, y la forma de acceder a ella dentro de una función

```
[3]:   aprox_pi = 3.141

[4]:   def acceso_global_pi():
      print('aprox_pi es llamada de acceso_global_pi co-
mo:',aprox_pi)


[5]:   acceso_global_pi()

[5]:   aprox_pi es llamada de acceso_global_pi como: 3.141
```

En una función no es posible modificar una variable global, al intentar hacer esto, Python crea una nueva variable, pero de alcance local.

```
[6]:   def intento_modificar_global_pi():
    aprox_pi=3.1416
    print('Modificación de aprox_pi como:',aprox_pi)


[7]:   intento_modificar_acceso_global_pi()

[7]:   Modificación de aprox_pi como: 3.1416

[8]:   aprox_pi

[8]:   3.141
```

Esto comprueba que el valor de la variable global `aprox_pi`, sigue siendo el mismo.

4.7 El módulo de matemáticas de Python

El módulo math de Python contienen funciones que permiten realizar diferentes tipos de cálculos típicos de matemáticas. Por ejemplo, trataremos de calcular el logaritmo base 10, de 1000, empleando las funciones ya definidas de Python.

```
[1]:  log10(1000)
```

```
---------------------------------------------------------
NameError                  Traceback (most recent call last)
<ipython-input-15-075cf7343366> in <module>
----> 1 log10(1000)

NameError: name 'log10' is not defined
```

La función definida de Python para calcular logaritmos base 10 es `log10()`. Sin embargo, el error que observas se debe a que la librería math de Python, no está cargada de manera predefinida.

Para resolver esto importa primero la librería

```
[2]:  import math
```

Para invocar a la función `log10`, utiliza la siguiente sintaxis

```
nombre_librería.función(argumento)
```

```
[3]:  math.log10(1000)

      3.0
```

Algunas funciones en la librería math de Python son

Para revisar más información y más funciones puedes revisar la documentación de Python:

https://docs.python.org/3/library/math.html

Así como existe un módulo de matemáticas, hay uno bastante útil y cuya librería contiene funciones para cálculos estadísticos.

4.8 Medidas de Tendencia Central y de Dispersión

A continuación, se utilizan algunas funciones de la librería de estadística de Python

Ejemplo 3.

Un estudiante ha tenido el siguiente registro de calificaciones (de 0 a 100) y desea conocer como ha sido su rendimiento calculando algunos estadísticos descriptivos como la media, la mediana, la moda, el rango, la desviación estándar y la varianza. Las calificaciones son:

Calificaciones: 68, 90, 80, 100, 80, 75, 85, 95, 70, 70
Este ejemplo está incluido en la carpeta del capítulo cuatro.

```
# ejemplo3_C4.py
""" Medidas de tendencia central y de dispersión."""

import statistics as estad

#Calificaciones

calificaciones= [68, 90, 80, 100, 80, 75, 85, 95, 70, 70]

# Medidas de tendencia central
media=estad.mean(calificaciones)
mediana=estad.median(calificaciones)
moda=estad.mode(calificaciones)

# Medidas de dispersión
rango=max(calificaciones)-min(calificaciones)
varianza=estad.pvariance(calificaciones)
desv_estandar=estad.pstdev(calificaciones)

print('Las medidas de tendencia central son:')
print(f'Media= {media:>7}')
print(f'Mediana= {mediana:>5}')
print(f'Moda= {moda:>6}')

print('Las medidas de dispersión son:')
print(f'Rango= {rango:>2}')
print(f'Varianza= {varianza:>0.3f}')
print(f'Desv. Estándar= {desv_estandar:>0.3f}')
```

```
Las medidas de tendencia central son:
Media=    81.3
Mediana=  80.0
Moda=     80
Las medidas de dispersión son:
```

```
Rango= 32
Varianza= 110.210
Desv. Estándar= 10.498
```

Como la media y la mediana no son muy diferentes, es posible afirmar que los datos tienen una buena medida de tendencia central de 80 aproximadamente. La desviación estándar es aproximadamente 10 por lo que a 10 unidades de la media se encuentran aproximadamente el 68% de las calificaciones según el Teorema de Chebyshev. El comportamiento, podemos intuir

C A P Í T U L O 5

Listas y Tuplas

5.1 Introducción

En esta sección aprenderás a crear y manipular listas.

En términos matemáticos una lista es una matriz o un vector. En términos de programación es un conjunto de elementos que contienen datos. Estos datos pueden ser homogéneos, es decir comparten ciertas similitudes; o heterogéneos, es decir tienen características diferentes.

Así que ponte listo y camina despacio, verás como el paisaje de la programación comienza a pintarse de colores

5.2 Listas

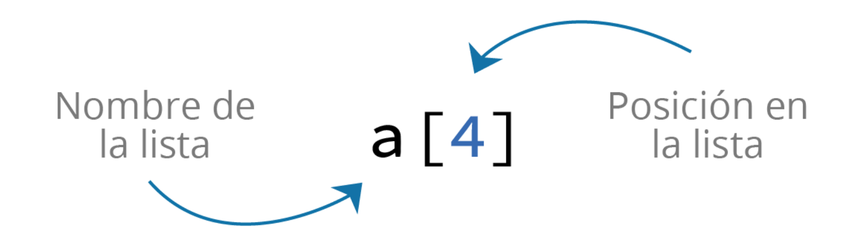
Una lista regularmente está compuesta por datos del mismo tipo o datos homogéneos sin embargo esto no es una regla

```
[1]: A = ['Patricia','México','F',1989]
[2]: A
[2]: ['Patricia','México','M',1989]
```

La lista anterior contiene 4 elementos, nombre, país, género y año de nacimiento. A diferencia de listas más comunes cómo:

```
[3]: a = [9,10,8.5,8,8,7.5,10]
[4]: a
[4]: [9,10,8.5,8,8,7.5,10]
```

Para llamar a cada elemento puedes utilizar la siguiente sintaxis



El primer elemento de la lista tendrá índice cero así,

```
[5]: a[0]
[5]: 9
[6]: a[3]
[6]: 8
[7]: A[3]
[7]: 1989
```

Para llamar el número de elementos en la lista,

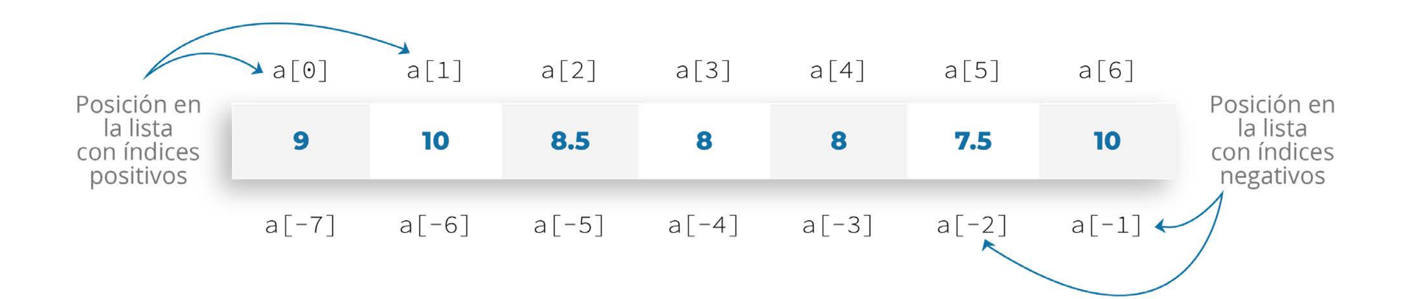
```
[8]: len(A)
[8]: 4
[9]: len(a)
[9]: 7
```

utiliza la función `len` (length).

La lista también puede llamarse empleando índices negativos,

```
[10]: A[-1]
[10]: 1989
[11]: a[-5]
[11]: 8.5
```

de manera que el elemento `a[2]` también puede ser invocado cómo `a[-5]`.



Los índices siempre deben ser enteros o expresiones enteras.

```
[12]: ax=1
[13]: ay = 4
[14]: a[ax-ay]
[14]: 8
```

Si es necesario cambiar algún elemento.

```
[15]: a[3] = 7.2
[16]: a
[16]: [9,10,8.5,7.2,8,7.5,10]
```

Esto, no es posible con las listas que contienen caracteres. Como en este ejemplo.

```
[17]: b = 'Python'
[18]: b[5]
[18]: 'n'
[19]: b[5]='m'
```

```
[19]:---------------------------------------------------
TypeError Traceback                (most recent call last)
<ipython-input-20-ed79499faf30> in <module>
----> 1 b[5]='m'
TypeError: 'str' object does not support item assignment
```

Si el elemento tiene un índice que no está en la lista, puede provocar un error.

```
[20]:  a[7]
```

```
-----------------------------------------------------
[20]: IndexError Traceback         (most recent call last)
<ipython-input-22-9cf13ba20553> in <module>
----> 1 a[7]
IndexError: list index out of range
```

Cada elemento puede utilizarse como una variable.

```
[21]: 5*a[2]+a[3]-a[6]/5
[21]: 47.7
```

Es posible añadir elementos a una lista vacía. Python puede realizar esta tarea de forma dinámica.

```
[22]: mi_secuencia=[]
[23]: for número in range(-2,3):
          mi_secuencia += [número]
[24]: mi_ secuencia
[24]: [-2, -1, 0, 1, 2]
```

En la sentencia `for`, la tarea que realiza `mi_secuencia` es crear un elemento en una lista y la anexa a `mi_secuencia`. Esta forma de trabajar de Python permite agregar cualquier tipo de elementos a una lista. Por ejemplo:

```
[25]: letras = []
[26]: letras += 'Porejemplo'
[27]: letras
[27]: ['P','o','r','e','j','e','m','p','l','o']
```

Concatenar dos listas, significa dos o más cadenas en una sola. Esta tarea es posible en Python con el operador +.

```
[28]: Lista = A + a
[29]: Lista
[29]: ['Patricia','México','F',1989,9,10,8.5,7.2,8,7.5,10]
```

El operador + regresa una lista nueva con los elementos de lista A seguidos de la lista a. Para visualizar los índices y sus valores, puedes utilizar el siguiente código.

```
[30]: for i in range(len(Lista)):
          print(f'{i}:{Lista[i]}')
[30]:  0:Patricia
       1:México
       2:F
       3:1989
       4:9
       5:10
       6:8.5
       7:7.2
       8:8
       9:7.5
       10:10
```

Y como antes, puedes acceder a los elementos de la lista a través de sus índices.

5.3 Tuplas

Las tuplas son secuencias de datos, pero, inmutables y con mucha frecuencia contienen datos heterogéneos. La longitud de una tupla es el número de elementos que contiene, y a diferencia de una lista, este número de elementos no puede cambiar en la ejecución de un programa.
Para crear una tupla vacía, utiliza paréntesis en lugar de corchetes

```
[1]: datos_cliente1 = ()
[2]: datos_cliente1
[2]: ()
[3]: len(datos_cliente1)
[3]: 0
```


Para añadir elementos a la tupla

```
[4]: datos_cliente1='Paty','F','23'
[5]: datos_cliente1
[5]: ('Paty', 'F', '23')
```

Aunque los elementos de una tupla no se pueden iterar, si es posible llamar a ellos de manera individual. Como los índices de una tupla la lista inicia en cero por ejemplo:

```
[6]: print(datos_cliente1[0],'nació en el año de', 2021-int
(datos_cliente1[2]))
Paty nació en el año de 1998
```

También es posible concatenar tuplas para formar una nueva

```
[7]: complemento_info = ('Feb','1998')
[8]: datos_c1 = datos_cliente1 + complemento_info
[9]: datos_c1
[9]: ('Paty', 'F', '23', 'Feb', '1998')
```

Es posible acceder a cualquier secuencia de elementos, asignando cada elemento de la secuencia con una variable separada por comas. Aquí dos ejemplos de cómo hacerlo.

```
[10]: datos_paciente1 = ('Paco',[1.80,69])
[11]: nombre,estatura_peso = datos_paciente1
[12]: nombre
[12]: 'Paco'
[13]: estatura_peso
[13]: [1.8,69]
```

Si en la tupla solo hay caracteres.

```
[14]: primera,segunda = 'Va'
[15]: print(f'{primera},{segunda}')
V,a
```

Si son valores.

```
[16]: Estatura,Peso,Temperatura = (1.8,69,37.5)
[17]: print(f'{Estatura},{Peso},{Temperatura}')
1.8,69,37.5
```

También se puede acceder a un valor al indexarlo. La función enumerate recibe una lista o una tupla que contiene dos elementos un índice y un valor. Respectivamente, crea una lista o una tupla.

```
[18]: estudiantes=['Hugo','Paco','Luis']
[19]: list(enumerate(estudiantes))
```

```
[19]:  [(0, 'Hugo'),(1, 'Paco'),(2, 'Luis')]
[20]:  tuple(enumerate(estudiantes))
[20]:  ((0, 'Hugo'),(1, 'Paco'),(2, 'Luis'))
```

Una forma de presentar estos valores.

```
[21]: for index,valor in enumerate(estudiantes):
         print(f'{index}:{valor}')
      0: Hugo
      1: Paco
      2: Luis
```

5.4 Separando secuencias

Es posible seleccionar elementos de una secuencia o sucesión. Es decir, seleccionar subconjuntos de una sucesión que sea mutable por ejemplo:

```
[1]: potencias2 = [2,4,8,16,32,64,128,264]
[2]: potencias2[3:5]
[2]: [16, 32]
```

Observa que la anotación indica que de la lista potencias2, se seleccionan los elementos con índices 3 al 5, sin incluir el 5. Esta notación permite extraer elementos de potencias2 de formas variadas, es decir:

```
[3]: potencias2[:5]
[3]: [2, 4, 8, 16, 32]
```

Selecciona los elementos con índices de 0 a 4, ya que como antes, el elemento 5 no se incluye.

```
[4]: potencias2[3:]
[4]: [16, 32, 64, 128, 264]
```

Observa que esta selección recoge los valores con índices mayores o iguales a 4. Al omitir los índices

```
[5]: potencias2[:]
[5]: [2, 4, 8, 16, 32, 64, 128, 264]
```

se realiza una copia simple de la sucesión original.

También es posible seleccionar elementos no consecutivos añadiendo un tamaño de paso.

```
[6]: potencias2[::3]
[6]: [2, 16, 128]
```

O incluso es posible hacerlo con índices negativos.


```
[7]: potencias2[::-3]
[7]: [264, 32, 4]
```

Se puede modificar una lista cambiando los valores de una parte de los elementos y dejar al resto sin cambiar.

```
[8]: potencias2[0:3] = ['dos','cuatro','ocho']
[9]: potencias2
[9]: ['dos', 'cuatro', 'ocho', 16, 32, 64, 128, 264]
```

Otras variantes para modificar una lista son

```
[10]: potencias2[0:2] = []
[11]: potencias2[:]
[11]: ['ocho', 16, 32, 64, 128, 264]
```

Observa que se borran los primeros dos elementos. El siguiente código cambia los elementos con un tamaño de paso específico.

```
[12]: potencias2 = [2,4,8,16,32,64,128,264]
[13]: potencias2[::3] = [1,1,1]
[14]: potencias2
[14]: [1, 4, 8, 1, 32, 64, 1, 264]
[15]: id(potencias2)
[15]: 2374936891776
```

El snippet [15] pide el identificador de la lista. Este identificador es único y se preserva a pesar de realizar diferentes operaciones. Por ejemplo, al dejar la lista vacía.

```
[16]: potencias2[:] = []
[17]: potencias2
[17]: []
[18]: id(potencias2)
[18]: 2374936891776
```

Es posible observar que el identificador no cambia a pesar de cambiar los elementos en la lista. Al hacer la asignación

```
[19]: potencias2 = []
[20]: potencias2
[20]: []
[21]: id(potencias2)
[21]: 2374937250688
```

en el snippet [19], observa que se han borrado los elementos, por lo que se ha creado una nueva lista. Esta característica marca una diferencia esencial entre dejar una lista vacía y asignar una lista vacía.

Para borrar cualquier elemento de una lista es posible utilizar la sentencia `del`.

```
[22]: multiplos10 = list(range(0,100,10))
[23]: multiplos10
[23]: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
[24]: del multiplos10[3:6]
[25]: multiplos10
[25]: [0, 10, 20, 60, 70, 80, 90]
```

Para borrar con un tamaño de paso.

```
[26]: multiplos10 = list(range(0,100,10))
[27]: multiplos10
[27]: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
[28]: del multiplos10[::2]
[29]: multiplos10
[29]: [10, 30, 50, 70, 90]
```

O para borrar una lista completa.

```
[30]: del multiplos10[:]
[31]: multiplos10
[31]: []
```

5.5 Listas ordenadas

Una de las tareas más comunes en cómputo es ordenar los valores o elementos de una lista. La forma de ejecutar esta instrucción en Python, es con la instrucción sort.

```
[1]: edad = [65,26,28,36,18]
[2]: edad.sort()
[3]: edad
[3]: [18, 26, 28, 36, 65]
```

Aquí se ordenan los datos en orden ascendente. Si lo quieres hacer en orden descendente

```
[4]: edad.sort(reverse = True)
[5]: edad
[5]: [65, 36, 28, 26, 18]
```

Otra función de Python para ordenar es `sorted`. La función sorted regresa una lista nueva con los elementos ordenados. Con este comando la sucesión original no cambia.

Para ordenar una lista


```
[6]: edad = [65,26,28,36,18]
[7]: edades_ascendentes = sorted(edad)
[8]: edades_ascendentes
[8]: [18, 26, 28, 36, 65]
[9]: edad
[9]: [65, 26, 28, 36, 18]
```

Para ordenar una cadena de caracteres.

```
[10]: nombre = 'FRANCISCO'
[11]: letras_ascendentes = sorted(nombre)
[12]: letras_ascendentes
[12]: ['A', 'C', 'C', 'F', 'I', 'N', 'O', 'R', 'S']
[13]: nombre
[13]: 'FRANCISCO'
```

También se puede ordenar una tupla.

```
[14]: estudiantes = ('Hugo','Paco','Luis')
[15]: estud_ordenados = sorted(estudiantes)
[16]: estud_ordenados
[16]: ['Hugo', 'Luis', 'Paco']
[17]: estudiantes
[17]: ('Hugo', 'Paco', 'Luis')
```

Otro de los tópicos clásicos en cómputo es la búsqueda. Python realiza una búsqueda a través del índice de los elementos de una lista o una tupla.

```
[18]: edad = [65,12,19,80,36,28,16,18,36,16,45,32,36,59]
[19]: edad.index(45)
[19]: 10
```

La función `index`, indica el lugar o índice en donde se encuentre el valor que se requiere encontrar. Un error del tipo `ValueError` ocurre si el valor no está en la lista de búsqueda.

```
[20]: edad.index(43)
```

```
[20]: ---------------------------------------------------
ValueError                Traceback (most recent call last)
<ipython-input-21-3d7a28478f51> in <module>
----> 1 edad.index(43)
ValueError: 43 is not in list
```

Una forma de verificar si algún valor está en la lista es con el comando in.

```
[21]: 56 in edad
[21]: False
```

```
[22]: 36 in edad
[22]: True
```

Otras funciones aplicables a las listas son insertar o borrar elementos. Enseguida se muestran algunos ejemplos.

Para insertar un elemento con un índice específico, puedes utilizar la función insert.

```
[23]: vegetales = ['espinaca','apio']
[24]: vegetales.insert(0,'lechuga')
[25]: vegetales
[25]: ['lechuga', 'espinaca', 'apio']
```

Para agregar elementos al final de la lista, puedes utilizar la función append.

```
[26]: vegetales.append('esparrago')
[27]: vegetales
[27]: ['lechuga', 'espinaca', 'apio', 'esparrago']
```

Para añadir más elementos al final de la lista es posible utilizar el comando extend.

```
[28]: Precio = (10,15,8,30)
[29]: vegetales.extend(precio)
[30]: vegetales
[30]:  ['lechuga', 'espinaca', 'apio', 'esparrago', 10, 15, 8, 30]
```

Prácticamente esta última tarea es equivalente a concatenar dos listas.

La función `remove`, borra el primer elemento con un valor específico. Un error del tipo `ValueError` ocurre si el elemento no está en la lista

```
[31]: vegetales.remove('esparrago')
[32]: vegetales
[32]:  ['lechuga', 'espinaca', 'apio', 'esparrago']
```

Para dejar una lista vacía es posible utilizar `clear`,

```
[33]: vegetales.clear()
[34]: vegetales
[34]: []
```

que es equivalente a

```
vegetales[:]=[]
```

La función count determina el número de veces que aparece un elemento en una lista

```
[35]: edades = [15, 21, 16, 19, 20, 16, 19, 15, 21, 17, 18,
```



```
24, 20, 24, 21, 19, 18, 16, 19, 16]
[36]: for edad in range(15,25):
          print(f'La edad {edad} se repite {edades.count(edad)} veces')
[36]: La edad 15 se repite 2 veces
      La edad 16 se repite 4 veces
      La edad 17 se repite 1 veces
      La edad 18 se repite 2 veces
      La edad 19 se repite 4 veces
      La edad 20 se repite 2 veces
      La edad 21 se repite 3 veces
      La edad 22 se repite 0 veces
      La edad 23 se repite 0 veces
      La edad 24 se repite 2 veces
```

La función reverse invierte el orden de una lista. A deferencia de sort, que tratamos anteriormente, reverse no crea una lista nueva.

```
[37]: vegetales = ['lechuga','espinaca','apio','pepinillo']
[38]: vegetales.reverse()
[39]: vegetales
[39]: ['pepinillo', 'apio', 'espinaca', 'lechuga']
```

La función copy, copia los elementos de una lista en una nueva.

```
[40]: copia_vegetales = vegetales.copy()
[41]: copia_vegetales
[41]: ['pepinillo', 'apio', 'espinaca', 'lechuga']
```

Esta tarea es equivalente a

```
copia_vegetales=vegetales[:]
```

5.6 Comprensión de listas

La comprensión de listas es una funcionalidad de Python que permite generar listas nuevas, este es un estilo simplista y elegante de Python aquí algunos ejemplos.

Para anexar elementos a una lista, habías utilizado este código:

```
[1]: lista1 = []
[2]: for dato in range (1,11):
         lista1.append(dato)
[3]: lista1
[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Pero con una comprensión de lista.

```
[4]: lista2=[dato for dato in range(1,11)]
[5]: lista2
[5]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Observa que solo se necesita una línea para sustituir el ciclo for. Para cada elemento `dato` la comprensión evalúa la expresión a la izquierda y le asigna una lista nueva. más aún la instrucción del snippet [4] puede ser reemplazada por

```
[6]: lista3 = list(range(1,11))
[7]: lista3
[7]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Es posible realizar tareas dentro de la comprensión de lista, regularmente cálculos. A esta tarea se le llama mapear. Mapear debes entenderlo como llevar cosas (elementos de una lista) de un lado a otro (a otra lista) Por ejemplo, los cuadrados de los primeros 7 números naturales.

```
[8]: cuad10=[dato*dato for dato in range(1,11)]
[9]: cuad10
[9]: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

En términos de mapeos, estás mapeado (llevando) una lista [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] a otra lista [1, 4, 9, 16, 25, 36, 49, 64, 81, 100].

Después, es posible seleccionar elementos que tengan alguna característica deseable, a esta operación se le llama filtrar y es posible realizarla utilizando una sentencia if. Aquí se muestra la lista anterior, pero únicamente con los elementos menores o iguales a 50.

```
[10]: cuad = [dato*dato for dato in range(1,11) if dato*dato<=50]
[11]: cuad
[11]: [1, 4, 9, 16, 25, 36, 49]
```

Y si es de interés aquellas potencias de números pares

```
[12]: cuad2 = [dato*dato for dato in range(1,11) if dato % 2 ==0]
[13]: cuad2
[13]: [4, 16, 36, 64, 100]
```

La sentencia for puede procesar cualquier iterable. El siguiente es un ejemplo de la versatilidad de Python para manipular listas.

```
[14]: vegetales = ['apio','brocoli','lechuga']
[15]: vegetales2 = [dato.upper() for dato in vegetales]
[16]: vegetales2
[16]: ['APIO', 'BROCOLI', 'LECHUGA']
```

Observa que la lista inicial tenía a sus elementos escritos con minúsculas y en la comprensión de lista, asigna estos elementos a otra lista, pero escritos con mayúsculas.

Además de la comprensión de listas, Python tiene otra funcionalidad que le da ventaja sobre otros lenguajes al manipular grandes bases de datos. Esta funcionalidad es llamada **generador de expresiones.** Este generador tiene cierta similitud con la comprensión de listas, pero al final genera objetos iterables. Esto es conocido como evaluación débil. La comprensión de listas realiza una evaluación robusta, por lo que crea otra lista inmediatamente después de la ejecución. Para una cantidad considerable de datos esto puede ocasionar un uso considerable de tiempo y memoria, por lo que el general el generador de expresiones reduce el consumo de recursos.

Por ejemplo, este código general los cuadrados pares de una lista

```
[17]: lista = [-3,5,8,4,1,-6,7,10]
[18]: for valor in (x**2for x in lista if x%2==0):
          print(valor,end=' ')
[18]: 64 16 36 100
```

Si hacemos esto con un **generador de expresiones**

```
[19]: cuadrados_pares = (x**2 for x in lista if x%2==0)
[20]: cuadrados_pares
[20]: <generator object <genexpr> at 0x00000142EA4AE660>
```

El "`<generator object <genexpr>`" indica que cuadrados_pares es un objeto generador y que fue creado por un generador de expresiones. Observa que no muestra los valores, esto es una consecuencia de la evaluación débil del generador. Esto contribuye a no utilizar recursos de memoria y tiempo de la computadora. Si quieres observar los resultados debes solicitarlo, por ejemplo:

```
[21]: list(cuadrados_pares)
[21]: [64, 16, 36 100]
```

Hasta aquí has visto algunas características funcionales de Python como comprensión de listas, filtraje y mapeo. Ahora utilizaremos funciones específicas qué hacen esta tarea como `filter` y `map`.

El siguiente código identifica los números pares de una lista de enteros.

```
[22]: lista=[-3,5,8,4,1,-6,7,10]
[23]: def pares(x):
          """Regresa el valor de la lista solo si es un número par"""
          return x % 2==0
[24]: list(filter(pares,lista))
[24]: [8, 4, -6, 10]
```

Aquí la función `filter` tiene como primer argumento la función `pares.` La función `pares` también tiene un argumento y regresa `True` si su argumento es par. La función `filter` llama entonces a la función `pares` para cada valor (de la lista) en el segundo argumento.
La función `filter` regresa un iterador (una lista generada) por lo que los resultados de `filter` no se producen hasta que iteras sobre ellos (la función `list` los manda llamar), este es otro ejemplo de una evaluación débil.

Una forma más dinámica de generar la misma lista, es con una comprensión, esto es

```
[25]: [valor for valor in lista if pares(valor)]
[25]: [8, 4, -6, 10]
```

Recuerda que este puede ser un código más corto, pero que puede tener ciertas desventajas en tiempo de cómputo y uso de memoria cuando las listas son muy largas.

Algunas veces es necesario utilizar funciones que realizan tareas relativamente sencillas, como en el ejemplo anterior la función `pares`. Para estas tareas Python permite utilizar una función a la que llama `lambda` y que prácticamente sirve para lo que sea. Esta función es como la χ que usas en álgebra. Por ejemplo

```
[26]: list(filter(lambda x:x%2==0,lista))
[26]: [8, 4, -6, 10]
```

Una función `lambda` es reconocida como una función *anónima* o *sin nombre* y es útil para llamar alguna función. Aquí están las similitudes entre la sintaxis de una función `def`

```
def nombre_función(parámetros
    return expresión
```

y una función `lambda`

```
lambda parámetros:expresión
```

Esto facilita la tarea de buscar el nombre específico para alguna función, que va a realizar alguna tarea sencilla.

Utilizaremos una lambda para introducir la función map.

```
[27]: lista
[27]: [-3, 5, 8, 4, 1, -6, 7, 10]
[28]: list(map(lambda x:x**3,lista))
[28]: [-27, 125, 512, 64, 1, -216, 343, 1000]
```

La función `map` tiene dos argumentos. El primero recibe un valor y la forma en que lo regresará, el segundo es una lista de valores iterables. La función `map` también realiza una evaluación débil, por lo que al pasarlos por la función `list` es posible ver los valores.

La misma lista la podemos generar con una comprensión como se muestra a continuación.

```
[29]: [valor**3 for valor in lista]
[29]: [-27, 125, 512, 64, 1, -216, 343, 1000]
```

Sí de la lista anterior sólo queremos los números pares es posible combinar las funciones map y filter

```
[30]: list(map(lambda x:x**3,filter(lambda x:x%2==0,lista)))
[30]: [512, 64, -216, 1000]
```


Naturalmente con una comprensión la formulación es más compacta

```
[31]: [x**3 for x in lista if x % 2 == 0]
[31]: [512, 64, -216, 1000]
```

En las secciones anteriores has comparado valores, algunas veces para ordenar o para encontrar máximos o mínimos. Sin embargo, estas comparaciones también se pueden llevar a cabo con objetos más complejos como cadenas de caracteres, por ejemplo:

```
[32]: 'Pato'<'murciélago'
[32]: True
```

Este resultado puede interpretarse como *verdadero*, debido a que la `P` está ubicada después de `m` en el abecedario, sin embargo, las cadenas son comparadas por los valores numéricos de sus caracteres. En este caso las letras minúsculas tienen valores numéricos más altos que las letras mayúsculas. Para revisar el valor numérico de las letras es posible utilizar la función `ord.`

```
[33]: ord('P')
[33]: 80
[34]: ord('m')
[34]: 109
[35]: ord('M')
[35]: 77
```

Si hubieras escrito murciélago con mayúsculas

```
[36]: 'Pato'<'Murciélago'
[36]: False
```

Ahora la cadena pato tiene un valor numérico mayor a murciélago.

Considera la siguiente lista de animales.

```
animales=['Pato','murciélago','Jirafa','gato','Ratón','conejo']
```

Si los quieres ordenar alfabéticamente, entonces el orden debe ser

```
'conejo', 'gato', 'Jirafa', 'murciélago', 'Pato', 'Ratón'
```

Observa que, en términos de orden, conejo es el mínimo en la lista, y ratón es el máximo. Para que Python pueda comprender este orden, es necesario que todas las palabras sean mayúsculas o minúsculas.

```
[37]: animales=['Pato','murciélago','Jirafa','gato','Ratón','conejo']
[38]: min(animales,key=lambda a:a.upper())
[38]: 'conejo'
[39]: max(animales,key=lambda a:a.upper())
[39]: 'Ratón'
```

La función key tiene como argumento una función de un parámetro que regresa otro valor. En este código lambda llama a la tarea upper (mayúsculas) para obtener la versión en mayúsculas de cada palabra, al final las funciones min y max comparan los valores numéricos de cada palabra.

Otra función que puede ser útil es zip, que permite iterar múltiples listas de datos de manera simultánea.

```
[40]: lugar = ['México','Colombia','Brasil']
[41]: ProdIB=[3.8,5.0,3.2]
[42]: for pais,pib in zip(lugar,ProdIB):
          print(f'País: {pais}, PIB= {pib}')
      País: México, PIB= 3.8
      País: Colombia, PIB= 5.0
      País: Brasil, PIB= 3.2
```

Aquí, el snippet [42] llama a la función zip para empaquetar la información en tuplas. Que luego se desempacan en pais y pib.

La función zip determina el número de tuplas, de acuerdo al argumento con menor número de elementos, en este caso las tuplas tienen el mismo número de elementos.

5.7 Listas de 2 dimensiones

Supongamos que un nutriólogo lleva a cabo la supervisión de 2 personas que siguen un régimen de control de peso. Después de 4 visitas bimestrales obtuvo la siguiente tabla de datos con respecto al peso de los pacientes en kilogramos.

	Bimestre 1	Bimestre 2	Bimestre 3	Bimestre 4
Paciente 1	118	117	114	110
Paciente 2	84.5	81.3	82	81

Para ingresar estos datos a Python es posible hacer una lista de 2 dimensiones.

```
[43]: peso = [[118,117,114,110], [84.5,81.3,82,81]]
```

Esto también se puede escribir como

```
peso = [[118,117,114,110],
        [84.5,81.3,82,81]]
```

Para poder visualizar los elementos de la lista con la forma de una matriz.


```
[44]:for fila in peso:
         for valor in fila:
             print(valor,end=' ')
         print()
     118 117 114 110
     84.5 81.3 82 81
```

La tabla para este ejercicio es posible representarla, en términos matemáticos, como una matriz. Una matriz es un arreglo numérico en filas y columnas. En términos de programación representa una lista de 2 dimensiones.

	Columna 1	Columna 2	Columna 3	Columna 4
Fila 1	118	117	114	110
Fila 2	84.5	81.3	82	81

Cada elemento de la lista tiene una posición específica interpretada por Python. Por ejemplo, el elemento peso[1][2], corresponde al elemento en la posición [1][2], es decir, fila 1, columna 2.

	Columna 1	Columna 2	Columna 3	Columna 4
Fila 1	peso[1][1]	peso[1][2]	peso[1][3]	peso[1][4]
Fila 2	pero[2][1]	peso[2][2]	peso[2][3]	peso[2][4]

5.8 Visualización de datos estáticos

Para este sección se utilizarán algunas herramientas de visualización que tiene Python y para ejemplificarlas empezaremos simulando cientos de miles de tiros de dado. Como sabrán en un dado hay seis caras y, por tanto, seis posibles resultados. Entonces, la probabilidad de que caiga como resultado el número 1 es de 1/6, la probabilidad de que caiga como resultado el número 2 es un 1/6, la probabilidad de que caiga como resultado el número 3 es de un 1/6 y así sucesivamente. Dicho lo anterior, veremos la tendencia de los tiros de dado por medio de gráficas de barras.

Empezaremos importando las librerías que vamos a necesitar:

```
timport matplotlib.pyplot as plt
import numpy as np
import random
import seaborn as sns
```

Después hagamos una lista con todos los resultados de tiros de dados y llamemosla “tirosdados”:

```
tirosdados = [random.randrange(1,7) for i in range(600)]
```

Entonces el ciclo `for` esta asegurando que se repita 600 veces, es decir, que generé resultados de tiros de dados que van del 1 al 6.

Posteriormente vamos a sacar un par de variables de la lista de “tirosdados”. Primero, obtendremos la variable “valores” que corresponde a los posibles resultados en los tiros, es decir, 1, 2, 3, 4, 5 y 6. Después obtendremos la variable “frecuencias” que corresponde a cuántas veces salió cada valor, por ejemplo, nos dirá cuántas veces salió el número 4 como resultado del tiro del dado.

Para construir las variables “valores” y “frecuencias” necesitaremos la función `unique()` que nos devolverá los valores únicos y la función `return_counts()` que conatrá cada uno de los resultados.

```
valores, frecuencias= np.unique(tirosdados, return_counts=-
True)
```

Ya con las variables por visualizar empecemos con la gráfica. Comencemos por declarar el título:

```
titulo = f ‘Resultados de tirar los dados {len(tirosdados)}
veces’
```

Ahora, definamos la ventana y los ejes:

```
sns.set_style(‘whitegrid’)
axes = sns.barplot(x=valores, y=frecuencias, palette =
‘bright’)
```

Para finalizar la gráfica de barras, pon título de ejes y otras etiqeutas importantes:

```
axes.set_title(titulo)
axes.set(xlabel=’Valores’, ylabel=’Frecuencias’)
```

Y si tienes creativo pon los número arriba de cada barra para que indique la frecuencia:

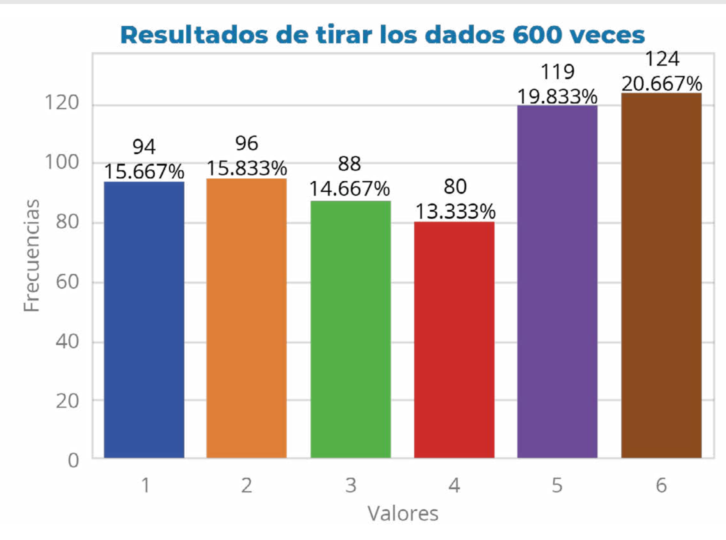
```
for bar, frecuencias in zip(axes.patches, frecuencias):
text_x=bar.get_x()+bar.get_width()/2.0
text_y=bar.get_height()
```



```
text= f’{frecuencias:,}\n{frecuencias/len(tirosdados):.3%}’
axes.text(text_x, text_y, text, fontsize=11, ha=’center’, va=’bo-
ttom’)
```

Recuerda trabajar en una sola celda para que sea más sencillo correr todos los comando, adjunto el código listo:

```
tirosdados = [random.randrange(1,7) for i in range(600)]
valores, frecuencias= np.unique(tirosdados, return_counts=-
True)
titulo = f'Resultados de tirar los dados {len(tirosdados)} ve-
ces'
sns.set_style('whitegrid')
axes = sns.barplot(x=valores, y=frecuencias, palette = 'bri-
ght')
axes.set_title(titulo)
axes.set(xlabel='Valores', ylabel='Frecuencias')
axes.set_ylim(top=max(frecuencias)*1.10)
for bar, frecuencias in zip(axes.patches, frecuencias):
  text_x=bar.get_x()+bar.get_width()/2.0
  text_y=bar.get_height()
  text= f'{frecuencias:,}\n{frecuencias/len(tirosdados):.3%}'
  axes.text(text_x, text_y, text, fontsize=11, ha='center',
  va='bottom')
```



NOTA: Si se presentará un error acerca de las comillas (“ “) simplemente reemplaza el símbolo manualmente. Cuando se realiza copy/paste puede cambiar el carácter y ocasionar problemas.

Bien, aquí terminamos con esta lección. Recuerda repasar el código para comprender los comandos, entre más contenido entiendas menos habrá que memorizar.

CAPÍTULO 6

Diccionarios y conjuntos

6.1 Introducción

Anteriormente habías trabajado con cadenas, listas y tuplas, ahora trabajarás con colecciones no ordenadas de elementos únicos. En términos matemáticos aprenderás a crear conjuntos y a manipular las operaciones definidas para ellos.

"Acomoda tu silla de trabajo preferida, acerca el teclado a tus manos, ponte atento y comencemos a programar"

6.2 Dicccionarios

Un diccionario tiene es una colección no ordenada de parejas, estas parejas contienen llaves y datos. Los datos pueden ser números (enteros, flotantes), cadenas (strings, booleanos) o arreglos (listas,tuplas) e incluso otro diccionario.

Es posible modificar los datos de un diccionario, o su longitud. Es decir, los diccionarios son objetos mutables, sin embargo, las llaves asociadas a los datos, no lo son.

Para crear un diccionario puedes utilizar llaves e ingresar sus elementos separados con una coma. Cada elemento se escribe con la siguiente notación:

Llave : valor

El siguiente código crea un diccionario de los usuarios de una caja registradora en una tienda de autoservicio, en donde cada persona tiene asignado un nickname. Observa que es posible cambiar el nickname de las personas, pero no su nombre.

```
[1]: usuario_nickname = {'Laura':'la','Daniel':'dn','Alberto':'al',
                         'Rogelio':'ro'}
[2]: usuario_nickname
[2]: {'Laura': 'la','Daniel': 'dn','Alberto': 'al','Rogelio': 'ro'}
```

Los diccionarios, por definición, son listas no ordenadas, por lo que es posible que al ejecutar el código no aparezca exactamente el mismo orden en el que fue escrito.

Para conocer la cantidad de parejas contenidas en el diccionario

```
[3]: len(usuario_nickname)
[3]: 4
```

El siguiente diccionario mapea, los nombres de los usuarios con su edad. Observa que las mismas llaves ahora tienen diferentes valores.

```
[4]: usuario_edad={'Laura':28,'Daniel':31,'Alberto':27,'Rogelio':65}
[5]: usuario_edad
[5]: {'Laura': 28, 'Daniel': 31, 'Alberto': 27, 'Rogelio': 65}
```

Para generar un arreglo con los nombres de los usuarios y su edad podemos utilizar:

```
[6]: for usuario,edad in usuario_edad.items():
         print(f'{usuario} tiene {edad} años cumplidos')

     Laura tiene 28 años cumplidos
     Daniel tiene 31 años cumplidos
     Alberto tiene 27 años cumplidos
     Rogelio tiene 65 años cumplidos
```

Es posible acceder a los valores a través de sus llaves

```
[7]:  usuario_edad['Daniel']
[7]:  31
[8]:  usuario_edad['Rogelio']
[8]:  65
```

Y también es posible cambiar el valor de alguna de las llaves. Supongamos que la edad de Rogelio fue mal capturada. Para corregirla.

```
[9]:  usuario_edad['Rogelio']=35
[10]: usuario_edad
[10]: {'Laura': 28, 'Daniel': 31, 'Alberto': 27, 'Rogelio': 35}
```


Si requieres añadir un nuevo elemento al diccionario

```
[11]: usuario_edad['Eva']=29
[12]: usuario_edad
[12]: {'Laura': 28, 'Daniel': 31, 'Alberto': 27, 'Rogelio': 35, 'Eva': 29}
```

Supongamos que Daniel cambia se traslada a otro departamento. Para borrarlo del diccionario.

```
[13]: del usuario_edad['Daniel']
[14]: usuario_edad
[14]: {'Laura': 28, 'Alberto': 27, 'Rogelio': 35, 'Eva': 29}
```

Si buscas en el diccionario una llave que no está incluida, Python te lo hará saber con un mensaje de error.

```
[15]:  usuario_edad['Pepe']
```

```
[15]: ---------------------------------------------------------
KeyError                  Traceback (most recent call last)
<ipython-input-28-58b28069d714> in <module>
----> 1 usuario_edad['Pepe']

KeyError: 'Pepe'
```

El método get de los diccionarios permite conocer el valor asociado a alguna llave.

```
[16]:  usuario_edad['Eva']
[16]:  29
```

Es posible prevenir el error que muestra Python con el usuario 'Pepe'al utilizar este método.

```
[17]:  usuario_edad.get['Pepe']
```

Sin embargo, al ejecutar el código observarás que no muestra nada. Para hacer este método útil, puedes hacer una sencilla modificación.

```
[18]:  usuario_edad.get['Pepe', 'Pepe no está incluido']
[18]:  'Pepe no está incluido'
```

Otra posibilidad para resolver este inconveniente es verificar si la llave está en el diccionario.

```
[19]: 'Pepe' in usuario_edad
[19]: False
[20]: 'Pepe' not in usuario_edad
[20]: True
[21]: 'Laura' in usuario_edad
[21]: True
```

Los diccionarios tienen sus propios métodos para iterar (manipular) el contenido. Esto puede ser, a través de sus llaves(keys), o sus valores(values).

```
[22]: usuario_edad={'Laura':28,'Daniel':31,'Alberto':27,
      'Rogelio':65}
[23]: for usuario in usuario_edad.keys():
          print(usuario, end=' ')
```

```
Laura Daniel Alberto Rogelio
```

```
[24]: for edad in usuario_edad.values():
          print(edad, end=' ')
```

```
28 31 27 65
```

Tanto keys, como values, solo muestran una vista del diccionario, por ejemplo:

```
[25]:  usuario_edad={'Laura':28,'Daniel':31,'Alberto':27,
       'Rogelio':65}
[26]:  usuarios=usuario_edad.keys()
[27]:  for llave in usuarios:
           print(llave, end=' ')
```

```
Laura Daniel Alberto Rogelio
```

Si añadimos otro elemento al diccionario (llave y valor).

```
[28]:  usuario_edad['Eva']=29
```

Verificaremos que key solo actualiza al diccionario, pero no conserva una copia de la información original.

```
[29]: for usuario in usuario_edad.keys():
          print(usuario, end=' ')
```

```
Laura Daniel Alberto Rogelio Eva
```

Sin embargo, esta vista no conserva la misma información cuando se modifican los datos, es decir, los métodos `keys` y `values` no realizan una copia de los datos en `usuario_edad.`

En algunas ocasiones es necesario trabajar con las llaves y los valores del diccionario. Esto es posible haciendo del mismo, una lista que contenga tanto las llaves como los valores. Esto es posible con la función `list.` Esta tarea no modifica al correspondiente diccionario.

```
[30]: list(usuario_edad.keys())
[30]: ['Laura', 'Daniel', 'Alberto', 'Rogelio', 'Eva']
[31]: list(usuario_edad.values())
[31]: [28 31 27 65 29]
```



```
[32]: list(usuario_edad.items())
[32]: [('Laura', 28), ('Daniel', 31), ('Alberto', 27),
       ('Rogelio', 65), ('Eva', 29)]
```

Para ordenar en orden alfabético las llaves del diccionario.

```
[33]: for usuario in sorted(usuario_edad.keys()):
          print(usuario, end=' ')
```

```
Alberto Daniel Eva Laura Rogelio
```

Python permite comparar diccionarios. Observa el siguiente ejemplo en el que se muestra los apellidos y nombres de tres personas que teóricamente asisten a un evento.

```
[34]: asistentes_dia1={'Ramírez':'Laura','Cortés':
      'Daniel','Pérez':'Alberto'}
[35]: asistentes_dia2={'García':'Rogelio','Ramírez':
      'Laura','Pérez':'Alberto'}
[36]:  asistentes_dia3={'Pérez':'Alberto','Cortés':
      'Daniel','Ramírez':'Laura'}
[37]:  asistentes_dia1 == asistentes_dia2
[37]:  False
[38]:  asistentes_dia2 != asistentes_dia3
[38]:  True
[39]:  asistentes_dia3 == asistentes_dia1
[39]:  True
```

Observa que en los snippets [37], [39], se trata de verificar si los diccionarios son iguales. Como los diccionarios son colecciones no ordenadas de elementos, el orden no representa un factor que indique a Python que los diccionarios son diferentes.

Ejemplo 1

En este ejemplo se calculan las ventas promedio en $, de 3 agentes o promotores de productos alimenticios en una semana. Aquí las llaves son los nombres de los agentes. Estas llaves mapean el nombre, a una lista de valores que representa las ventas que realizaron.

El ciclo `for`, separa los contenidos del diccionario en dos iterables: agente y ventas. En la línea 12, la función sum, suma los valores de la lista venta, mientras que, en la línea 13, la suma total se divide entre la cantidad de elementos de la lista venta.

Por otro lado, en el mismo ciclo, en la línea 14 se suman todas las ventas realizadas, mientras que en la línea 15 se calcula cada iteración la cantidad de ventas logradas por todos los agentes. En la última línea se imprime el promedio de ventas de todos los involucrados.

Este ejemplo está incluido en la carpeta del capítulo seis.

```python
# ejemplo1_Cap6
"""Cálculo de ventas por agente"""
ventas_agentes={
    'Laura': [12000, 10500, 9800, 11100],
    'Daniel': [11000, 7800, 7200, 6500],
    'Alberto': [11200, 9500, 10800, 10100]
}
venta_total=0
num_ventas=0

for agente,venta in ventas_agentes.items():
    total=sum(venta)
    print(f'{agente} tuvo en promedio, ${total/len(venta):0.3f}')
    venta_total += total
    num_ventas += len(venta)

    print(f'Las ventas promedio de la agencia fue de
    {venta_total/num_ventas:0.3f}')
```

```
'Cálculo de ventas por agente'
Laura tuvo en promedio, $10850.000
Daniel tuvo en promedio, $8125.000
Alberto tuvo en promedio, $10400.000
Las ventas promedio de la agencia fue de $9791.667
```

Ejemplo 2

Para contar el número de palabras en un texto, es posible construir un diccionario que realice esta tarea. En este ejemplo las líneas 4, 5 y 6 crean una cadena de texto que es separada en palabras. Aunque los espacios en blanco concatenan a la cadena. estos espacios se sustituyen por un dato reconocible por Python que es ignorado en este proceso. A esta tarea se le reconoce en inglés como *tokenizing* a *string*. En la línea 8 se crea un diccionario vacío.

Las llaves del diccionario son las palabras únicas, y sus valores son las veces que esta llave se repite.

Este ejemplo está incluido en la carpeta del capítulo seis.

```python
1   # ejemplo2_Cap6
2   """Conteo de palabras"""
3
4   cuento=('cuenta un cuento que una princesa se la pasa '
5           'cuenta que cuenta ese cuento llamado la princesa '
6           'que cuenta')
7
8   palabras_contadas={}
9
10  # para contar las palabras que no se repiten
11  for palabra in cuento.split():
12      if palabra in palabras_contadas:
13       palabras_contadas[palabra] +=1
14   else:
15       palabras_contadas[palabra]=1
16
17  print(f'{"PALABRA":<12} REPETICIONES')
18  for palabra,veces in sorted(palabras_contadas.items()):
19      print(f'{palabra:<12}{veces}')
20
21  print('\nNúmero de palabras únicas en el texto:',len(pa-
labras_contadas))
```

```
PALABRA     REPETICIONES
cuenta      4
cuento      2
ese         1
la          2
llamado     1
pasa        1
princesa    2
que         3
se          1
un          1
una         1
Número de palabras únicas en el texto: 11
```

En la línea 10 *tokeniza* el texto, con el método `split`, el cual separa las palabras utilizando como argumento delimitador una palabra, si no se escribe ningún argumento, `split` usa por default, un espacio en blanco. El método regresa una lista de tokens Para cada palabra, la línea 12 determina si la palabra ya fue contada. Si esto es verdadero incrementa en 1 al contador. Si es falso, es la primera vez que se identifica la palabra y le asigna 1 a su contador.

Las líneas 17 a la 21 se encargan de mostrar los resultados: el número de repeticiones de cada palabra y el número de palabras únicas.

La tarea de contar palabras puede volverse una tarea común, en algunos casos. Por lo que Python tiene la herramienta Counter, que permite realizar esta tarea. Esta función se encuentra en las librerías del módulo collections. El programa anterior se puede simplificar con esta función, como se muestra enseguida.

```python
[40]:  from collections import Counter
[41]:  cuento=('cuenta un cuento que una princesa se la pasa '
       'cuenta que cuenta ese cuento llamado la princesa '
```

```
         'que cuenta')
[42]:  contador=Counter(cuento.split())
[43]:  for palabra,veces in sorted(contador.items()):
        print(f'{palabra:<12}{veces}')

cuenta      4
cuento      2
ese         1
la          2
llamado     1
pasa        1
princesa    2
que         3
se          1
un          1
una         1

[44]:  print('\nNúmero de palabras únicas en el texto:',len(-
contador.keys()))

Número de palabras únicas en el texto: 11
```

Cuando sea necesario actualizar algún diccionario, por ejemplo, renovar el nickname de los usuarios de una caja registradora.

```
[45]:  usuario_id={}
```

Esto genera un diccionario vacío, al cual le añadiremos el nombre y nickname de un usuario.

```
[46]:  usuario_id.update({'Laura':'lau'})
[47]:  usuario_id
[47]:  {'Laura': 'lau'}
```

Esta misma tarea se puede realizar utilizando automáticamente el nombre como un parámetro asociado con su respectivo valor

```
[48]:  usuario_id.update({'Daniel':'dani'})
[49]:  usuario_id
[49]:  {'Laura': 'lau', 'Daniel': 'dani'}
```

Para corregir los valores asignados a las llaves, se pueden actualizar los datos utilizando la misma estrategia.

```
[50]:  usuario_id.update({'Daniel':'dan'})
[51]:  usuario_id
[51]:  {'Laura': 'lau', 'Daniel': 'dan'}
```


Esto se debe a que la llave no cambia, como se había mencionado anteriormente.

6.3 Comprensión de diccionarios

La comprensión de diccionarios, como la comprensión de listas, proporciona una notación más compacta para generar diccionarios, además de mapear diccionarios en diccionarios. Como el siguiente ejemplo en donde se intercambian llaves y valores

```
[1]: usuario_edad={'Laura':28,'Daniel':31,'Alberto':27,
     'Rogelio':65}
[2]:  edad_usuario={edad:usuario  for  usuario,edad  in  usuario_
edad.items()}
[3]:   edad_usuario
[3]:   {28: 'Laura', 31: 'Daniel', 27: 'Alberto', 65: 'Rogelio'}
```

Como en los diccionarios, la comprensión de diccionarios está delimitada por los símbolos de agrupación `{}`. En esta comprensión se iteran los pares usuario-edad a través de `usuario_edad.items()` para signarle la llave y el valor a un nuevo diccionario con pares `edad-usuario.`

Una comprensión de diccionario también puede mapear valores en nuevos valores. Observa este ejemplo.

```
[4]: ventas_agentes= {'Laura': [12000, 10500, 9800, 11100],
                      'Daniel': [11000, 7800, 7200, 6500],
                      'Alberto': [11200, 9500, 10800, 10100]}
[5]: ventas_promedio={k:sum(v)/len(v) for k,v in ventas_agentes.items()}
[6]: ventas_promedio
[6]: {'Laura': 10850.0, 'Daniel': 8125.0, 'Alberto': 10400.0}
```

6.4 Conjuntos (Sets)

Un conjunto es una colección no ordenada de valores únicos. Los conjuntos pueden contener objetos no mutables como cadenas, enteros, números de tipo flotante y tuplas que contienen elementos no mutables.

Aunque los conjuntos son iterables, no es posible indexar sus elementos y como consecuencia no es posible obtener partes de un conjunto como en el caso de las listas.

El código siguiente, genera un conjunto llamado animales.

```
[1]: animales={'gato','ratón','perro','garza','perico','perro','pato'}
[2]: animales
[3]: {'garza', 'gato', 'pato', 'perico', 'perro', 'ratón'}
```

Observa que en el snippet [1], el elemento `'perro'` está escrito dos veces, pero al solicitar los elementos del conjunto, solo aparece una vez. Recuerda que los elementos del conjunto son únicos, por lo que no admiten duplicidades de elementos.

Para determinar la cantidad de elementos en el conjunto.

```
[4]:  len(animales)
[4]:  6
```

Si quieres verificar si algún elemento está incluido en el conjunto

```
[5]:  'perro' in animales
[5]:  True
[6]:  'gallo' in animales
[6]:  False
[7]:  'gallo' not in animales
[7]:  True
```

Los conjuntos son iterables, esta característica permite procesar cada elemento en un ciclo for.

```
[8]: for animal in animales:
     print(animal.upper(),end=' ')
     PERICO GATO GARZA PERRO RATÓN PATO
```

Los conjuntos son colecciones no ordenadas de elementos, por lo que el orden, en este caso, es irrelevante.

Es posible crear un conjunto, a partir de otras colecciones de valores utilizando la función set. El código siguiente genera una lista con elementos pares en diferentes rangos y luego los concatena.

```
[9]:  pares=list(range(0,15,2))+list(range(8,29,2))
[10]: pares
[10]: [0, 2, 4, 6, 8, 10, 12, 14, 8, 10, 12, 14, 16, 18, 20]
```

Para hacer un conjunto con esta lista.

```
[11]: set(pares)
[11]: {0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20}
```

Observa que se eliminan los elementos repetidos. Puedes comparar este conjunto con el siguiente,

```
[12]: pares2=list(range(0,28,2))
[13]: pares2
[13]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26]
[14]: set(pares) == set(pares2)
[14]: False
```

El operador == verifica si los conjuntos son exactamente iguales. Si lo son, la expresión es evaluada como verdadera, si no lo son, será falsa. Otras formas de comparar conjuntos.


```
[15]: set(pares) > set(pares2)
[15]: False
[16]: set(pares) < set(pares2)
[16]: True
```

Los operadores >, < verifican si un conjunto es subconjunto propio del otro. Es decir, para dos conjuntos A, B, diremos que A es un subconjunto propio de B, en código: A< B, si todos los elementos de A están en B, pero existe al menos un elemento en B que no está en A.

```
[17]: {2,4,6,8} < {4,8,6,2}
[17]: False
```

Otra forma de verificar si un conjunto es subconjunto propio de otro es

```
[18]: {2,5,8}.issubset({5,9,8,2})
[18]: True
```

De manera similar. A es un superconjunto propio de B, y en código lo denotaremos como A>B, si todos los elementos de B están en A, pero existe al menos un elemento en A que no está en B

```
[19]: {2,4,6,8}>{4,8,6,2}
[19]: False
[20]: {5,9,8,2}>{2,5,8}
[20]: True
```

Para verificar si un conjunto es un superconjunto impropio de otro

```
[21]: {2,4,6,8}>={4,8,6,2}
[21]: True
[22]: {2,4,6,8,10}>{4,8,6,2}
[22]: True
[23]: {2,8}>{4,8,6,2}
[23]: False
```

Hay 5 operaciones que es posible realizar con conjuntos

Unión de conjuntos.

La unión de dos conjuntos, es un conjunto que tiene los elementos únicos de ambos conjuntos. Esta operación es posible con el operador &, o escribiendo el método tipo `union`.

```
[24]: {'Laura','Daniel'} | {'Rogelio','Eva','Laura'}
[24]: {'Daniel', 'Eva', 'Laura', 'Rogelio'}
[25]: {2,4,6,8}|{1,4,6,8,9}
[25]: {1, 2, 4, 6, 8, 9}
[26]: {2,4,6}.union([1,2,3,4,5])
[26]: {1, 2, 3, 4, 5, 6}
```

Intersección de conjuntos

La intersección de dos conjuntos, es el conjunto que contiene a los elementos únicos, que tienen en común los dos conjuntos. Esta operación es posible con el operador &, o escribiendo el método tipo `intersection`.

```
[24]: {'Laura','Daniel'} & {'Rogelio','Eva','Laura'}
[24]: {'Laura'}
[25]: {2,4,6,8}&{1,4,6,8,9}
[25]: {4, 6, 8}
[26]: {1,3}&{2,4,6}
[26]: set()
```

En este último ejemplo, no hay intersección por lo que Python regresa un conjunto vacío.

```
[26]: {2,4,6}.intersection([1,2,3,4,5])
[26]: {2, 4}
```

Diferencia de conjuntos

La diferencia entre dos conjuntos A, B, se denota A-B y se define como el conjunto cuyos elementos están en A pero no están en B

```
[24]: {'Laura','Daniel'} - {'Rogelio','Eva','Laura'}
[24]: {'Daniel'}
[25]: {2,4,6,8} - {1,4,6,8,9}
[25]: {2}
[26]: {1,3} - {2,4,6}
[26]: {1,3}
[26]: {2,4,6}.difference([1,2,3,4,5])
[26]: {6}
```

Diferencia simétrica de conjuntos

La diferencia simétrica entre dos conjuntos A, B, se denota A^B y se define como el conjunto formado por lo elementos que están en A, o en B, pero no en ambos a la vez.

```
[24]: {'Laura','Daniel'} ^ {'Rogelio','Eva','Laura'}
[24]: {'Daniel','Rogelio','Eva'}
[25]: {2,4,6,8} - {1,4,6,8,9}
[25]: {1,2,9}
[26]: {1,3} - {2,4,6}
[26]: {1,2,3,4,6}
```



```
[26]: {2,4,6}.symmetric_difference([1,2,3,4,5])
[26]: {1,3,5,6}
```

Conjuntos disjuntos

Dos conjuntos son disjuntos si no tienen elementos en común. Puedes verificar si dos conjuntos son disjuntos utilizando el método tipo `isdisjoint`.

```
[26]: {1,3}.isdisjoint({2,4,6})
[26]: True
[26]: {1,2,3}.isdisjoint({2,4,6})
[26]: False
```

Las anteriores operaciones entre conjuntos generan conjuntos nuevos. Sin embargo, es posible realizar estas operaciones y actualizar el conjunto original, sin la necesidad de crear un conjunto nuevo. Esto se logra con el operador seguido de la asignación =. Aquí un ejemplo.

```
[26]: numeros = {2,4,6}
[26]: impares = {3,5,7}
[26]: numeros |= impares #operador unión seguido de =
[26]: numeros
[26]: {2, 3, 4, 5, 6, 7}
```

Observa que el conjunto `numeros` original se modificó al finalizar la operación.

Esto también ocurre con el método tipo `update` cuando quieres realizar la operación unión con el conjunto que está utilizando.

```
[26]: numeros.update(range(5,10))
[26]: numeros
[26]: {2, 3, 4, 5, 6, 7, 8, 9}
```

El método tipo `add` permite añadir elementos al conjunto. Si el argumento no está presente en el conjunto, lo añade. Si está, el conjunto no se modifica.

```
[26]: numeros.add(11)
[26]: numeros.add(7)
[26]: numeros
[26]: {2, 3, 4, 5, 6, 7, 8, 9, 11}
```

El método `remove`, quita el elemento que este en el argumento. Si este elemento no se encuentra en el conjunto, se produce un error tipo `KeyError`.

```
[26]: numeros.remove(9)
[26]: numeros
[26]: {2, 3, 4, 5, 6, 7, 8, 11}
```

El método `discard` también puede quitar elementos al conjunto, pero no provoca un error si el elemento no se encuentra presente.

Por último, una función que puede ser útil es el método tipo `clear`. Este método deja vacío al conjunto que se está invocando.

```
[26]: numeros.clear())
[26]: numeros
[26]: set()
```

CAPÍTULO 7

Programación con Numpy

7.1 Introducción

En este capítulo aprenderás a utilizar algunas cualidades básicas de **NumPy** (Numerical Python). NumPy es una de las librerías más populares de Python, ya que puede procesar listas de múltiples dimensiones que incluyen lazos anidados, o comprensión de listas con diversas sentencias `for`. En la parte final, realizarás una introducción a la librería pandas que es útil para el análisis en ciencia de datos.

Así es que,...

"Ponte zapatos cómodos, ropa holgada y fresca. Sigamos caminando por este sendero, orientados por la brújula de la programación"

7.2 Arreglos de Datos

Como en la sección anterior, invocaremos a la librería NumPy como `np`

```
[1]: import numpy as np
```

El módulo numpy proporciona diversas formas de crear arreglos. Por ejemplo, la función array es útil para este propósito

```
[2]: factoriales = np.array([1,2,6,24,120,720])
```

Esta función copia el contenido de su argumento y lo coloca dentro del array. Para verificar el tipo de objeto que se genera

```
[3]:  type(factoriales)
[4]:  factoriales
[4]:  array([  1,   2,   6, 24, 120, 720])
```

Observa que, en la salida de factoriales, NumPy separa cada valor del siguiente con una coma y un espacio con alineación a la derecha. NumPy deja tantos lugares como sean necesarios, para que cada elemento cuente con el mismo campo, aunque en algunos casos, sean espacios en blanco. El ancho del campo queda determinado por el número que tenga la mayor cantidad de caracteres.

La función `array` copia automáticamente las dimensiones del argumento. Por ejemplo, el siguiente arreglo (array) está formado por tres filas y dos columnas.

```
[5]: np.array([[3,4],[6,7],[10,11]])
[5]: array([[ 3,  4],
            [ 6,  7],
            [10, 11]])
```

Un arreglo tiene atributos que poco a poco irás conociendo. En esta sección utilizaremos los siguientes.

```
[6]: import numpy as np
[7]: enteros = np.array([[9,10,11,12],[8,7,6,5]])
[8]: enteros
[8]: array([[ 9, 10, 11, 12],
           [ 8,  7,  6,  5]])
[9]:  flotantes=np.array([[9**(1/2),10**(1/2),3.33,],[10/3,3.4,3.03]])
[10]: flotantes
[10]: array([[3.        , 3.16227766, 3.33      ],
            [3.33333333, 3.4       , 3.03      ]])
```

Puedes verificar el tipo de elemento generado con `dtype`.

```
[11]:  enteros.dtype
[11]:  dtype('int32') # en algunas plataformas puede ser int64
[12]:  flotantes.dtype
[12]:  dtype('float64')
```

El atributo `ndim` contiene el número de dimensiones del arreglo y el atributo `shape` contiene una tupla que especifica las dimensiones del arreglo.

```
[13]:  enteros.ndim
[13]:  2
[14]:  flotantes.ndim
[14]:  2
[15]:  enteros.shape
[15]:  (2, 4)
```



```
[16]:  flotantes.shape
[16]:  (2, 3)
```

Para conocer el número de elementos en el arreglo puedes utilizar el atributo `size`. Y el número de bytes necesarios para almacenarlos se puede conocer mediante `itemsize`.

```
[17]:  enteros.size
[17]:  8
[18]:  enteros.itemsize
[18]:  4
[19]:  flotantes.size
[19]:  6
[20]:  flotantes.itemsize
[20]:  8
```

Para manipular arreglos es posible hacerlo con el estilo funcional de programación, pero también se puede hacer empleando iteraciones externas, como hasta hoy lo has estado haciendo

```
[21]: for fila in flotantes:
          for columna in fila:
              print(columna, end=' ')
          print()

3.0   3.1622776601683795   3.33
3.3333333333333335   3.4   3.03
```

Puedes, por otro lado, convertir un arreglo multidimensional en uno, de una dimensión con el atributo `flat`.

```
[22]: for n in enteros.flat:
          print(n,end=' ')

      9 10 11 12 8 7 6 5
```

NumPy tiene funciones que permiten rellenar arreglos con valores específicos. Estas funciones son `zeros, ones` y `full`.

```
[23]: import numpy as np
[24]: np.zeros(5)
[24]: array([0., 0., 0., 0., 0.])
[25]: np.ones((3,2),dtype = int)
[25]: array([[1, 1],
             [1, 1],
             [1, 1]])
[26]: np.full((2,4),10)
[26]: array([[10, 10, 10, 10],
             [10, 10, 10, 10]])
```

Además de generar arreglos con ceros, unos y valores específicos, también es posible generar arreglos con rangos de valores específicos. La función `arange` permite crear arreglos con tamaño de paso entero.

```
[27]: import numpy as np
[28]: np.arange(8)
[28]: array([0, 1, 2, 3, 4, 5, 6, 7])
[29]: np.arange(3,8)
[29]: array([3, 4, 5, 6, 7])
[30]: np.arange(12,2,-3)
[30]: array([12,  9,  6,  3])
```

También es posible crear arreglos con tamaños de paso que no sea entero (float), con la función `linspace`.

```
[31]:  np.linspace(0.0,1.0,num =5)
[31]:  array([0.,0.25,0.5,0.75,1.])
```

Si tienes un arreglo de una dimensión y quieres armar un arreglo de dos dimensiones debes utilizar el método `reshape`. En este ejemplo se crea un arreglo de una dimensión con 20 elementos, para crear un arreglo de 4 filas y 5 columnas

```
[32]: np.arange(21,1,-1).reshape(4,5)
[32]: array([[21, 20, 19, 18, 17],
             [16, 15, 14, 13, 12],
             [11, 10,  9,  8,  7],
             [ 6,  5,  4,  3,  2]])
```

Si las dimensiones de los arreglos no están acopladas, puede resultar un mensaje de error.

```
[33]:  np.arange(0,1000,0.5).reshape(4,2000)
```

```
---------------------------------------------------------
ValueError                Traceback (most recent call last)
<ipython-input-49-ccd56e764913> in <module>
----> 1 np.arange(0,1000,0.5).reshape(4,2000)

ValueError: cannot reshape array of size 2000 into shape (4,2000)
```

Si el arreglo tiene 1,000 o más elementos, NumPy no muestra la totalidad de datos. Para no ocupar espacio la librería divide al arreglo permitiendo ver una parte de la información.

El siguiente ejemplo genera un arreglo con 10,000 elementos, para generar otro con 5 filas y 2,000 columnas.

```
[34]: np.arange(0,10000).reshape(5,2000)
[34]: array([[   0,    1,    2, ..., 1997, 1998, 1999],
             [2000, 2001, 2002, ..., 3997, 3998, 3999],
             [4000, 4001, 4002, ..., 5997, 5998, 5999],
             [6000, 6001, 6002, ..., 7997, 7998, 7999],
             [8000, 8001, 8002, ..., 9997, 9998, 9999]])
```


En este otro ejemplo, el mismo arreglo con 10,000 elementos, genera otro con 50 filas y 200 columnas.

```
[34]: np.arange(0,10000).reshape(50,200)

[34]: array([[   0,    1,    2, ...,  197,  198,  199],
             [ 200,  201,  202, ...,  397,  398,  399],
             [ 400,  401,  402, ...,  597,  598,  599],
             ...,
             [9400, 9401, 9402, ..., 9597, 9598, 9599],
             [9600, 9601, 9602, ..., 9797, 9798, 9799],
             [9800, 9801, 9802, ..., 9997, 9998, 9999]])
```

7.3 Operaciones con arreglos

NumPy posee una gran variedad de operadores que pueden aplicarse a los diferentes arreglos que puedes elaborar. Iniciaremos con algunas operaciones básicas que se aplican a cada valor del arreglo.

```
[1]:  import numpy as np
[2]:  pares = np.arange(2,15,2)
[3]:  pares
[3]:  array([ 2,  4,  6,  8, 10, 12, 14])
[4]:  pares+1
[4]:  array([ 3,  5,  7,  9, 11, 13, 15])
[5]:  pares*3
[5]:  array([ 6, 12, 18, 24, 30, 36, 42])
[6]:  pares**(1/2)
[6]:  array([1.41421356, 2.        , 2.44948974, 2.82842712,
             3.16227766, 3.46410162, 3.74165739])
[7]:  pares
[7]:  array([ 2,  4,  6,  8, 10, 12, 14])
```

Este último snippet, permite verificar que el arreglo original no cambió. Otras funcionalidades de NumPy si cambian al arreglo, por ejemplo, `+=`, o `-=`.

```
[8]:  pares -=1
[9]:  pares
[9]:  array([ 1,  3,  5,  7,  9, 11, 13])
```

Puedes realizar operaciones entre arreglos que tengan las mismas dimensiones.

```
[10]: lista1 = np.arange(2,18,3)
[11]: lista1
```

```
[11]: array([ 2,  5,  8, 11, 14, 17])
[12]: lista2 = np.linspace(-2,20,6)
[13]: lista2
[13]: array([-2. ,  2.4,  6.8, 11.2, 15.6, 20. ])
[14]: lista2-lista1
[14]: array([-4. , -2.6, -1.2,  0.2,  1.6,  3. ])
[15]: lista2/lista1
[15]: array([-1.          ,  0.48        ,  0.85        ,
1.01818182,  1.11428571, 1.17647059])
```

También puedes comparar arreglos

```
[16]: lista1<lista2
[16]: array([False, False, False, True, True, True])
```

Numpy realiza una comparación individual entre los elementos respectivos de cada arreglo

```
[17]: lista2 >= 4
[17]: array([False, False, True, True, True, True])
[18]: lista1 == lista2
[18]: array([False, False, False, False, False, False])
```

Los arreglos tienen varios métodos que pueden realizar cálculos con sus contenidos. Por default, estos métodos ignoran las dimensiones del arreglo y utilizan a todos los elementos para hacer las operaciones. Por ejemplo, el siguiente arreglo contiene las ventas de una pequeña tienda de autoservicio.

```
[19]: import numpy as np
[20]: ventas=np.array( [[554,606,710,851],[1244,898,416,1763],
                       [841,655,1105,1067]])
[21]: ventas
[21]: array([[ 554,  606,  710,  851],
             [1244,  898,  416, 1763],
             [ 841,  655, 1105, 1067]])
```

Utilizaremos los métodos `sum, min, max, mean, std, var`. Que calculan respectivamente, la suma, el valor mínimo, el máximo, el promedio, la desviación estándar y la varianza con todos los datos.

```
[22]: ventas.sum()
[22]: 10710
[23]: ventas.min()
[23]: 416
[24]: ventas.max()
[24]: 1763
[25]: ventas.mean()
[25]: 892.5
[26]: ventas.std()
```



```
[26]: 350.56656524355924
[27]: ventas.var()
[27]: 122896.91666666667
```

Pero también es posible hacerlo por filas o por columnas. Por ejemplo, los promedios por columna

```
[28]: ventas.mean(axis=0)
[28]: array([ 879.66666667,  719.66666667,  743.66666667, 1227.   ])
```

Y los promedios por fila

```
[29]: ventas.mean(axis=1)
[29]: array([ 680.25, 1080.25,  917.  ])
```

NumPy tiene además un conjunto de funciones que son llamadas universales. Estas operaciones regresan un arreglo nuevo como resultado.

```
[30]: import numpy as np
[31]: lista1 = np.arange(0,8)
[32]: lista1
[32]: array([0, 1, 2, 3, 4, 5, 6, 7])
[33]: np.sqrt(lista1)
[33]: array([0.        , 1.        , 1.41421356, 1.73205081, 2.,
        2.23606798, 2.44948974, 2.64575131])
```

La función `sqrt` de NumPy regresa la raíz cuadrada de cada uno de los elementos del arreglo. Esto lo habías hecho anteriormente, pero utilizando la potencia 1/2.

```
[34]: lista2 = np.arange(3,11)
[35]: lista2
[35]: array([ 3,  4,  5,  6,  7,  8,  9, 10])
[36]: np.add(lista1,lista2)
[36]: array([ 3,  5,  7,  9, 11, 13, 15, 17])
```

Este último resultado es equivalente a la operación entre arreglos

`lista1 + lista2`

Como la suma, el producto tiene su propia función

```
[37]: np.multiply(lista2,2)
[37]: array([ 6,  8, 10, 12, 14, 16, 18, 20])
```

Existen otras formas de multiplicar. Pero primero cambiemos las dimensiones de `lista2` en un arreglo de 2 filas y 4 columnas

```
[38]: lista3 = lista2.reshape(2,4)
[39]: lista3
```

```
[39]: array([[ 3,  4,  5,  6],
             [ 7,  8,  9, 10]])
```

Un nuevo arreglo es

```
[40]: lista4 = np.array([2,-4,6,-8])
[41]: lista4
[41]: array([ 2, -4,  6, -8])
```

Entonces, la multiplicación de los arreglos `lista3` y `lista4` es

```
[42]: np.multiply(lista3,lista4)
[42]: array([[  6, -16,  30, -48],
             [ 14, -32,  54, -80]])
```

Observa que, con esta forma de multiplicar, `lista4` se multiplicó con cada fila de lista3.

NumPy tiene una lista grande de funciones universales, la puedes consultar en :

https://numpy.org/doc/stable/reference/ufuncs.html

7.4 Indexando y cortando arreglos

Para indexar los elementos de un arreglo, es posible utilizar la misma sintaxis empleada en listas y tuplas, pero con la significativa diferencia, que ahora el índice puede tener más argumentos. Observa los siguientes ejemplos, con los datos de la tienda de autoservicio de la sección anterior.

```
[1]: import numpy as np
[2]: ventas=np.array ([[ 554, 606,  710,  851],
                       [1244, 898,  416, 1763],
                       [ 841, 655, 1105, 1067]])
[3]: ventas
[3]: array([[ 554,  606,  710,  851],
            [1244,  898,  416, 1763],
            [ 841,  655, 1105, 1067]])
```

Recuerda que, en el caso de índices, Python enumera al primer índice como 0. Entonces, para ubicar el elemento `ventas[2,1]`, se encuentra en la fila con índice 2, y columna con índice 1.

	Columna 0	Columna 1	Columna 2	Columna 3
Fila 0	ventas[0,0]	ventas[0,1]	ventas[0,2]	ventas[0,3]
Fila 1	ventas[1,0]	ventas[1,1]	ventas[1,2]	ventas[1,3]
Fila 2	ventas[2,0]	ventas[2,1]	ventas[2,2]	ventas[2,3]

Así.

```
[4]:  ventas[2,1]
[4]:  655
[5]:  ventas[1,3]
[5]:  1763
[6]:  ventas[1]
[6]:  array([1244,  898,  416, 1763])
```

Con esta notación está indicando a NumPy que solo requieres la fila con índice 1. Para solicitar las filas con índices 0 y 1

```
[7]: ventas[0:2]
[7]: array([[ 554,  606,  710,  851],
            [1244,  898,  416, 1763]])
```

Recuerda que con los índices, Python ignora el superior en `ventas[0:2]`. Entonces, para solicitar las filas con índices 1 y 2.

```
[8]: ventas[1:3]
[8]: array([[1244,  898,  416, 1763],
            [ 841,  655, 1105, 1067]])
```

Si requieres solo los elementos de la primera columna

```
[9]: ventas[:,0]
[9]: array([ 554, 1244,  841])
```

Y de las columnas con índices 1 y 2

```
[10]: ventas[:,1:3]
[10]: array([[ 606,  710],
             [ 898,  416],
             [ 655, 1105]])
```

Pero si necesitas las columnas con índices 0 y 2

```
[11]: ventas[:,[0,2]]
[11]: array([[ 554,  710],
             [1244,  416],
             [ 841, 1105]])
```

Los arreglos que regresa NumPy de la forma *array([[argumentos]])*, son llamados *vistas(views) o copias superficiales (shallow copies)*. Otros métodos que muestran resultados pueden definirse con NumPy a través de `view`.

```
[12]: import numpy as np
[13]: impares = np.arange(3,18,2)
```

```
[14]: impares
[14]: array([ 3,  5,  7,  9, 11, 13, 15, 17])
[15]: impares2=impares.view()
[15]: impares2
[16]: array([ 3,  5,  7,  9, 11, 13, 15, 17])
```

Aunque son los mismos arreglos, `impares` e `impares2` son dos objetos diferentes para Python. Para revisar esto puedes usar la función `id`.

```
[17]: id(impares)
[17]: 1580433432048
[18]: id(impares2)
[18]: 1580433367568
```

Para mostrar que `impares2` tiene los mismos datos que impares, modificaremos un elemento en `impares`.

```
[19]: impares[3]*=10
[20]: impares
[20]: array([ 3,  5,  7,  90, 11, 13, 15, 17])
[21]: impares2
[21]: array([ 3,  5,  7,  90, 11, 13, 15, 17])
[22]: impares[3]/=10
[23]: impares
[23]: array([ 3,  5,  7,  9, 11, 13, 15, 17])
[24]: impares2
[24]: array([ 3,  5,  7,  9, 11, 13, 15, 17])
```

Esto verifica que los arreglos tienen los mismos elementos, aunque son identificados como objetos diferentes.

Aunque el método `view` separa arreglos como objetos, ocupan memoria para compartir datos con otros arreglos. Sin embargo, algunas veces es necesario crear copias independientes, a estas copias se les llama *Deep Copies* (copias profundas). Este detalle es esencial en tópicos de programación, como la programación multinúcleo.

El método `copy` para arreglos genera una copia profunda del arreglo original.

Para verificar lo anterior, el siguiente código genera un arreglo con números impares y luego aplica el método `copy`.

```
[25]: import numpy as np
[26]: impares = np.arange(3,18,2)
[27]: impares
[27]: array([ 3,  5,  7,  9, 11, 13, 15, 17])
[28]: impares2 = impares.copy()
[29]: impares2
[29]: array([ 3,  5,  7,  9, 11, 13, 15, 17])
```


Ahora modificaremos un elemento del arreglo impares

```
[30]: impares[1]*=100
[31]: impares
[31]: array([ 3,  500,  7,  9, 11, 13, 15, 17])
[32]: impares2
[32]: array([ 3,  5,  7,  9, 11, 13, 15, 17])
```

Observa que la copia no cambio, de manera que `impares` e `impares2` son dos arreglos independientes.

Otros métodos de NumPy para los arreglos son `reshape` y `resize`. Ambos permiten la interacción de arreglos de dos dimensiones con arreglos de una dimensión. `reshape` produce una vista (`view`) o copia superficial del arreglo original con nueva dimensión. Este método no modifica al arreglo original.

```
[33]: import numpy as np
[34]: ventas = np.array([[ 500, 600, 550, 800],
                         [1200, 800, 400,1000]])
[35]: ventas
[35]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
[36]: ventas.reshape(1,8)
[36]: array([[ 500,  600,  550,  800, 1200,  800,  400, 1000]])
[37]: ventas
[37]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
```

El método `resize` modifica la forma del arreglo original

```
[38]: ventas.resize(1,8)
[39]: ventas
[39]: array([[ 500,  600,  550,  800, 1200,  800,  400, 1000]])
```

Para el caso específico de cambiar un arreglo multidimensional, en otro de una dimensión, los métodos `flatten` y `ravel` realizan esta tarea.

```
[40]: ventas = np.array([[ 500, 600, 550, 800],
                         [1200, 800, 400,1000]])
[41]: ventas
[41]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
[42]: vector_ventas=ventas.flatten()
[42]: array([[ 500,  600,  550,  800,
              1200,  800,  400, 1000]])
[43]: ventas
[43]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
```

El método `flatten` genera una copia profunda del arreglo original. Para verificar que no comparte información, el siguiente código modifica un elemento en la copia.

```
[44]: vector_ventas[0]=1500
[45]: vector_ventas
[45]: array([[1500,  600,  550,  800, 1200,  800,  400, 1000]])
[46]: ventas
[46]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
```

El método `ravel` genera una vista (`view`) del arreglo original, por lo que comparten los mismos datos.

```
[47]: ventas
[47]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
[48]: reacomodar_ventas = ventas.ravel()
[49]: reacomodar_ventas
[49]: array([[ 500,  600,  550,  800, 1200,  800,  400, 1000]])
[50]: reacomodar_ventas[0] =1000
[51]: reacomodar_ventas
[51]: array([[1000,  600,  550,  800, 1200,  800,  400, 1000]])
[52]: ventas
[52]: array([[1000,  600,  550,  800],
             [1200,  800,  400, 1000]])
```

La transpuesta de un arreglo con *n* filas y *m* columnas es otro arreglo que resulta de cambiar las filas del primero en las columnas del segundo, de tal manera que el arreglo resultante tiene *m* filas y *n* columnas. Para hacer esta tarea Python utiliza el método `T` para generar la transpuesta de un arreglo.

```
[53]: ventas = np.array([[ 500, 600, 550, 800],
                         [1200, 800, 400,1000]])
[54]: ventas
[54]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
[55]: ventas.T
[55]: array([[ 500, 1200],
             [ 600,  800],
             [ 550,  400],
             [ 800, 1000]])
```

La transpuesta no modifica al arreglo original.

```
[56]: ventas
[56]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
```

Si requieres agregar más filas o columnas puedes realizarlo con las opciones `vstack` y `hstack`.

Con el arreglo anterior

```
[57]: ventas
[57]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
```

Para añadirle una fila, al final.

```
[58]: ventas2 = np.array([[1100,1000,950,1100]])
[59]: np.vstack((ventas,ventas2))
[59]: array([[1000,  600,  550,  800],
             [1200,  800,  400, 1000],
             [1100, 1000,  950, 1100]])
```

Estas opciones no cambian al arreglo original

```
[60]: ventas
[60]: array([[ 500,  600,  550,  800],
             [1200,  800,  400, 1000]])
```

Para añadir dos columnas a ventas

```
[61]: ventas3 = np.array([[1100,900],[800,900]])
[62]: np.hstack((ventas,ventas3))
[62]: array([[ 500,  600,  550,  800, 1100,  900],
             [1200,  800,  400, 1000,  800,  900]])
```

7.5 Pandas

Pandas es la librería más popular de Python para trabajar con datos de tipo homogéneo, a través de sus índices. En este apartado revisarás dos colecciones de librerías útiles para el tratamiento estadístico de datos: `Series` y `Dataframes`.

Series de Pandas permite trabajar con arreglos unidimensionales. Series utiliza arreglos con índices enteros que inician en 0. A continuación algunos ejemplos.

El siguiente código crea un arreglo con Series de Pandas. Este arreglo representa el tiempo en minutos que invierten en videojuegos 4 estudiantes.

```
[1]: import pandas as pd
[2]: tiempo_web = pd.Series([160,256,98,108])
[3]: tiempo_web
[3]: 0    160
     1    256
     2     98
```

```
        3    108
  dtype:  int64
```

Pandas muestra el resultado en dos columnas con los índices alineados a la izquierda y los valores alineados a la derecha. Series también muestra el tipo de datos al final del arreglo.

Para crear una serie con el mismo elemento

```
[4]: pd.Series(3.1416,range(5))
[4]: 0    3.1416
     1    3.1416
     2    3.1416
     3    3.1416
     4    3.1416
     dtype: float64
```

Si quieres tener acceso a algún elemento

```
[5]: tiempo_web[1]
[5]: 256
```

Series tiene métodos clásicos para determinar las estadísticas descriptivas de un conjunto de datos

```
 [6]: tiempo_web.count()
 [6]: 4
 [7]: tiempo_web.mean()
 [7]: 155.5
 [8]: tiempo_web.min()
 [8]: 98
 [9]: tiempo_web.max()
 [9]: 256
[10]: tiempo_web.std()
[10]: 72.30260484012085
```

Todas estas tareas se pueden solicitar con `describe`.

```
[11]: tiempo_web.describe()
[11]: count        4.000000
       mean      155.500000
        std       72.302605
        min       98.000000
        25%      105.500000
        50%      134.000000
        75%      184.000000
        max      256.000000
     dtype:      float64
```


Es posible personalizar los índices en Series, a través de `index.`

```
[12]: tiempo_web=pd.Series([160,256,98,108],index=
      ['Laura','Daniel','Alberto','Eva'])
[13]: tiempo_web
[13]: Laura      160
      Daniel     256
      Alberto     98
      Eva        108
      dtype: int64
```

También puede definir un arreglo Series con un diccionario.

```
[14]: tiempo_web=pd.Series({'Laura':160,'Daniel':256,
      'Alberto':98,'Eva':108})
[15]: tiempo_web
[15]: Laura      160
      Daniel     256
      Alberto     98
      Eva        108
      dtype:     int64
```

Llamar a los elementos requiere la misma notación que antes, pero con los índices personalizados.

```
[16]: tiempo_web['Eva']
[16]: 108
```

Cuando los índices son cadenas, Pandas añade los identificadores a los atributos de Series, de manera que es posible llamar a los elementos con la siguiente notación.

```
[17]:  tiempo_web.Alberto
[17]:  98
```

Otros atributos incorporados en Series es el tipo de dato(`dtype`), los valores(`values`) y los índices(`index`)

```
[18]: tiempo_web.dtype
[18]: dtype('int64')
[19]: tiempo_web.values
[19]: array([160, 256,  98, 108], dtype=int64)
[20]: tiempo_web.index
[20]: Index(['Laura', 'Daniel', 'Alberto', 'Eva'],
       dtype='object')
```

Si un arreglo tipo Series, es una cadena (string), puedes utilizar los atributos de la cadena, invocando a los métodos para strings de Python.

El siguiente arreglo contiene algunos ingredientes de una receta para hacer galletas.

```
[21]: ingredientes = pd.Series(['Leche','Mantequilla',
      'Harina','Azúcar'])
[22]: ingredientes
[22]: 0          Leche
      1    Mantequilla
      2         Harina
      3         Azúcar
      dtype: object
```

Para verificar si alguna letra está contenida en alguno de los elementos, el método `contain` para cadenas puede realizar esta tarea.

```
[23]: ingredientes.str.contains('a')
[23]: 0    False
      1     True
      2     True
      3     True
      dtype: bool
```

7.6 Dataframes

Un `DataFrame` es un arreglo de dos dimensiones. Y cada columna del DataFrame es un arreglo tipo Series, justo como en la sección anterior. Por lo tanto, todos los atributos de Series son aplicables al DataFrame.

El siguiente código genera un diccionario y a partir de él, se crea el DataFrame. Los datos representan el peso en kilos registrados por una nutrióloga, a lo largo de cuatro meses, de cuatro pacientes.

```
[1]: import pandas as pd
[2]: reg_peso = {'Vanesa':[68,67,66,65],'Kevin':[89,89,90,88],
               'Fernanda':[59,60,60,62],'Patricia':[70,68,67,65]}
[3]: peso = pd.DataFrame(reg_peso)
[4]: peso
[4]:       Vanesa    Kevin     Fernanda  Patricia
      0        68       89          59        70
      1        67       89          60        68
      2        66       90          60        67
      3        65       88          62        65
```

Para añadir índices personalizados

```
[5]:  peso.index = ['Mes 1','Mes 2','Mes 3','Mes 4']
[6]:  peso
[6]:          Vanesa    Kevin   Fernanda  Patricia
       Mes 1     68       89         59        70
```



```
              Mes 2     67        89          60         68
              Mes 3     66        90          60         67
              Mes 4     65        88          62         65
```

Es posible seleccionar la información de cualquier individuo.

```
[7]: peso['Fernanda']
[7]: Mes 1  59
     Mes 2  60
     Mes 3  60
     Mes 4  32
     Name: Fernanda, dtype: int64
```

Como en Series, con los índices en formato string, se pueden solicitar de manera alternativa los datos de cualquier paciente.

```
[8]: peso.Patricia
[8]: Mes 1  70
     Mes 2  68
     Mes 3  67
     Mes 4  65
     Name: Patricia, dtype: int64
```

Puedes acceder a una fila a través del atributo `loc` de DataFrame.

```
[9]: peso.loc['Mes 1']
[9]: Vanes      68
     Kevin      89
     Fernanda   59
     Patricia   70
     Name: Mes 1, dtype: int64
```

También es posible acceder a las filas a través de índices. Para esto necesitas el atributo `iloc`.

```
[10]: peso.iloc[1]
[10]: Vanesa    67
      Kevin     89
      Fernanda  60
      Patricia  68
      Name: Mes 2, dtype: int64
```

Si requieres parte del arreglo, por ejemplo, del mes 1 al mes 3.

```
[11]:  peso.loc['Mes 1':'Mes 3']
[11]:   Vanesa  Kevin  Fernanda  Patricia
 Mes 1     68      89        59        70
 Mes 2     67      89        60        68
```

```
        Mes 3      66    90    60    67
```

Observa que está incluido el índice Mes 3.

Cuando utilizas los índices numéricos con `iloc`, el índice superior no se incluye.

```
[12]: peso.loc[0:2]
[12]:          Vanesa    Kevin   Fernanda   Patricia
        Mes 1      68       89         59         70
        Mes 2      67       89         60         68
```

Otra posibilidad, es extraer filas específicas.

```
[13]: peso.loc[['Mes 1','Mes 3']]
[13]:           Vanesa   Kevin   Fernanda  Patricia
        Mes 1       68      89         59        70
        Mes 3       66      90         60        67
```

Si es con índices numéricos.

```
[14]: peso.loc[[0, 2]]
[14]:         Vanesa    Kevin   Fernanda     Patricia
        Mes 1     68       89         59           70
        Mes 3     66       90         60           67
```

Con meses específicos y pacientes específicos.

```
[15]: peso.loc['Mes 2':'Mes 3',['Vanesa','Patricia']]
[15]:           Vanesa    Patricia
        Mes 2       68          68
        Mes 3       66          67
```

Cuando la selección la haces con índices.

```
[16]: peso.iloc[[0, 2], 0:3]
[16]:        Vanesa    Kevin      Fernanda
        Mes 1     68       89            59
        Mes 3     66       90            60
```

Otra herramienta de Pandas es la indexación booleana. Por ejemplo, para encontrar los pesos mayores o iguales a 70kg.

```
[17]:  peso[peso>=70]
[17]:          Vanesa   Kevin   Fernanda   Patricia
        Mes 1      NaN      89        NaN         70
        Mes 2      NaN      89        NaN        NaN
        Mes 3      NaN      90        NaN        NaN
        Mes 4      NaN      88        NaN        NaN
```


Ahora, los pesos mayores a 65kg, pero menores a 80kg.

```
[18]: peso[(peso>65)&(peso<80)]
[18]:         Vanesa     Kevin    Fernanda     Patricia
        Mes 1     68.0       NaN         NaN         70.0
        Mes 2     67.0       NaN         NaN         68.0
        Mes 3     66.0       NaN         NaN         67.0
        Mes 4      NaN       NaN         NaN          NaN
```

Los atributos `at` e `iat` permiten llamar a un valor único del DataFrame. Como antes, `at`, usa los índices personalizados e `iat` utiliza los índices numéricos.

```
[19]: peso.at['Mes 3','Patricia']
[19]: 67
[20]: peso.iat[2,0]
[20]: 66
```

Este último peso corresponde al paciente en la fila con índice 2 (Mes 3) y columna con índice 0 (Vanesa).

Estos atributos también permiten modificar algún elemento del DataFrame. Supongamos que el peso de Kevin en el mes 4 debe ser de 85kg, y el Fernanda debe ser 59 en el mes 2 (índices [1,2]).

```
[21]: peso.at['Mes 4','Kevin']=85
[22]: peso.at['Mes 4','Kevin']
[22]: 85
[23]: peso.iat[1,2]=59
[24]: peso.iat[1,2]
[24]: 59
```

El DataFrame queda de la forma

```
[25]: peso
[25]:         Vanesa    Kevin    Fernanda    Patricia
        Mes 1     68       89          59          70
        Mes 2     67       89          59          68
        Mes 3     66       90          60          67
        Mes 4     65       85          62          65
```

Como en series, el Dataframe tiene el método describe para determinar las estadísticas descriptivas.

```
[26]: peso.describe()
[26]:         Vanesa        Kevin     Fernanda      Patricia
 count     4.000000     4.000000     4.000000     4.000000
 mean     66.500000    88.250000    60.000000    67.500000
 std       1.290994     2.217356     1.414214     2.081666
 min      65.000000    85.000000    59.000000    65.000000
```

```
25%       65.750000  88.000000  59.000000  66.500000
50%       66.500000  89.000000  59.500000  67.500000
75%       67.250000  89.250000  60.500000  68.500000
max       68.000000  90.000000  62.000000  70.000000
```

Pandas calcula las estadísticas descriptivas utilizando números en formato float empleando 6 decimales de precisión. Para cambiar la cantidad decimales en la presentación llama a la función `set_option`.

```
[27]: pd.set_option('precision',1)
[28]: peso.describe()
[28]:       Vanesa    Kevin     Fernanda  Patricia
 count         4.0      4.0          4.0       4.0
 mean         66.5     88.2         60.0      67.5
 std           1.3      2.2          1.4       2.1
 min          65.0     85.0         59.0      65.0
 25%          65.8     88.0         59.0      66.5
 50%          66.5     89.0         59.5      67.5
 75%          67.2     89.2         60.5      68.5
 max          68.0     90.0         62.0      70.0
```

Con esta cantidad de decimales, puedes llamar al arreglo de promedios en el DataFrame

```
[29]: peso.mean()
[29]: Vanesa    66.5
       Kevin     88.2
       Fernanda  60.0
       Patricia  67.5
      dtype: int64
```

En el DataFrame original los individuos estaban en filas y los meses en columnas. Para colocar a los individuos en las columnas, solo es necesario pedir la transpuesta del DataFrame

```
[30]: peso.T
[30]:            Mes 1   Mes 2   Mes 3   Mes 4
      Vanesa       68      67      66      65
      Kevin        89      89      90      85
      Fernanda     59      59      60      62
      Patricia     70      68      67      65
```

El siguiente código muestra los estadísticos por mes con la función `describe`.

```
[31]: peso.T.describe()
[31]:          Mes 1    Mes 2    Mes 3     Mes 4
       count     4.0      4.0      4.0       4.0
       mean     71.5     70.8     70.8      69.2
       std      12.6     12.8     13.2      10.6
       min      59.0     59.0     60.0      62.0
       25%      65.8     65.0     64.5      64.2
       50%      69.0     67.5     66.5      65.0
       75%      74.8     73.2     72.8      70.0
       max      89.0     89.0     90.0      85.0
```


El peso promedio por mes.

```
[32]: peso.T.mean()
[32]: Mes 1     71.5
      Mes 2     70.8
      Mes 3     70.8
      Mes 4     69.2
      dtype: float64
```

DataFrame también permite ordenar la información por filas o por columnas. El ordenamiento se realiza a través de los índices. A continuación, se ordena el DataFrame por filas, con los índices en orden descendente.

```
[33]:  peso.sort_index(ascending=False)
[33]:       Vanesa    Kevin     Fernanda  Patricia
 Mes 4      65    85    62    65
 Mes 3      66    90    60    67
 Mes 2      67    89    59    68
 Mes 1      68    89    59    70
```

Para ordenar las columnas en orden ascendente.

```
[34]: peso.sort_index(axis=1)
[34]:       Fernanda  Kevin  Patricia   Vanesa
     Mes 1        62     85        65       68
     Mes 2        60     90        67       67
     Mes 3        59     89        68       66
     Mes 4        59     89        70       65
```

Otra de las bondades del dataframe es ordenar los valores de acuerdo a alguna fila o columna específica. El siguiente código ordena de forma descendente los datos del DataFrame con respecto a los valores del Mes 1.

```
[35]: peso.sort_values(by='Mes 1', axis=1,ascending=False)
[35]        Kevin     Patricia  Vanesa    Fernanda
 Mes 1        89          70      68          59
 Mes 2        89          68      67          59
 Mes 3        90          67      66          60
 Mes 4        85          65      65          62
```

Con las mismas condiciones, pero utilizando la transpuesta

```
[36]: peso.T.sort_values('Mes 1',ascending=False)
[36]:      Mes 1    Mes 2    Mes 3    Mes 4
 Kevin     89       89       90       85
 Patricia  70       68       67       65
 Vanesa    68       67       66       65
 Fernanda  59       59       60       62
```

Finalmente, también es posible llamar a los valores ordenados del mes 1

```
[37]: peso.loc['Mes 1'].sort_values(ascending=False)
[37]: Kevin     89
      Patricia  70
      Vanesa    68
      Fernanda  59
```

CAPÍTULO 8

Cadenas

8.1 Introducción

Este capítulo está pensado en hacer un acercamiento a las cadenas (strings) de caracteres. Mucha de la información disponible, en las grandes bases de datos, está guardadas o definidas en archivos que contienen cadenas de información. Una de las grandes aplicaciones de Python es la identificación de patrones en cadenas de texto. En este capítulo revisarás algunos principios para posteriormente y esta es el área fundamental para lo que conoce como Procesamiento de Lenguaje Natural.

"Llego el momento de tomar una cápsula reconfortante de programación.
Venga, vamos a programar"

8.2 Dando formato a las cadenas.

Recordarás que en las sesiones anteriores utilizabas la opción `f-string` para dar formato a los resultados de salida. Por default Python presenta el resultado como una cadena, incluso aunque se especifique otro. Por ejemplo.

```
[1]:   f'{3.141592:0.3f}'
[1]:   '3.141'
```

Observa que Python atiende la instrucción del formato float, pero termina mostrando el resultado como una cadena.

Hay diferentes tipos de formato para los diferentes tipos de datos. Algunos ejemplos de estos tipos son:

Enteros, con el tipo de presentación d.

```
[2]:  f'{150:d}'
[2]:  '150'
```

Caracteres, con el tipo de presentación c.

```
[3]:  f'{66:c} {98:c}'
[3]:  B b
```

Python asigna el caracter entero con su caracter alfabético.

Cadenas (strings), con el tipo de presentación s.

```
[4]:  f'{"Este curso está de":s} {100}'
[4]:  'Este curso está de 100'
```

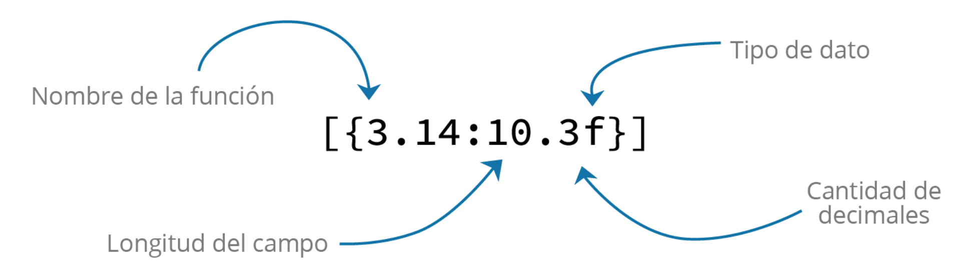
Recuerda que habías empleado el `f-string` para dar formato al texto de salida, y hasta podías especificar el número de caracteres para una cadena añadiendo espacios en blanco. Por default, Python da una alineación a la derecha, a los valores; y a las cadenas, a la izquierda. Además, para que esto ocurra debes encerrar los resultados entre corchetes.

Aquí hay algunos ejemplos que muestran la forma de asignar un campo en el `f-string`.

```
[5]: f'[{314:10d}]'
[5]: '[       314]'
[6]: f'[{"Alineación":15}{3.14:10f}]'
[6]: '[Alineación       3.140000]'
[7]: f'[{"Alineación":15}]'
[7]: '[Alineación     ]'
[8]: f'[{3.14:10f}]'
[8]: '[  3.140000]'
```

Python asigna a los números tipo float, 6 dígitos de precisión. Debido a esto, en la salida del snippet [6] y [7] aparece el número 3.140000.

La sintaxis que emplea float en el f-string es la siguiente:



Cuando no indicas la cantidad de decimales que requieres, Python añade los 6 lugares decimales por default.

Los números que aparecen después de los dos puntos (:) indican la longitud del campo. Esta longitud ya incluye la cantidad de caracteres de la cadena. Por ejemplo, en el snippet [7], la cadena Alineación ocupa 10 caracteres y añade otros 5 en forma de espacios en blanco.

Pero además de especificar la longitud del campo y la cantidad de decimales, también puedes alinear a la cadena con los símbolos > y <.

```
[9]:  f'[{"Alineación":12}{3.14:<10f}]'
[9]:  '[Alineación  3.140000  ]'
[10]: f'[{"Alineación":>15}]'
[10]: '[     Alineación]'
[11]: f'[{3.14:<10f}]'
[11]: '[3.140000  ]'
```

Y para centrar, tanto el texto como los valores

```
[12]:  f'[{"Alineación":^12}{3.14:^10f}]'
[12]:  '[ Alineación  3.140000 ]'
[13]:  f'[{"Alineación":^15}]'
[13]:  '[  Alineación   ]'
[14]:  f'[{3.14:^10.1f}]'
[14]:  '[   3.1    ]'
```

En el caso particular de cadenas de números, si requieres mostrar el signo. Puedes agregarlo en el f-string.

```
[15]: f'[{150:+10d}]'
[15]: '[      +150]'
```

Para agregar ceros en los espacios en blanco.

```
[16]: f'[{150:+010d}]'
[16]: '[+000000150]'
```

Regularmente a los números positivos no se les agrega el signo +. Pero, si requieres dejar el espacio en blanco para este tipo de números, utiliza un código como el siguiente.

```
[17]: print(f'{27:d}\n{27: d}\n{-27: d}')
[17]:  27
       27
      -27
```

Al operar con cantidades grandes, muchas veces es conveniente utilizar comas para indicar miles o millones. Esto es posible hacerlo con Python.

```
[18]:  f'{111222333:,d}'
[18]:  '111,222,333'
[19]:  f'{444555666.777:,.3f}'
[19]:  '444,555,666.777'
```

En versiones de Python anteriores a la 3.6, no era posible utilizar los f-strings. Para dar formato a las cadenas utilizaban el método `format` de Python. Aunque los f-string tienen muchas bondades, aquí se muestran algunas opciones que aún son útiles con el método `format`.

```
[20]:  '{:0.4f}'.format(3.141592)
[20]:  '3.1416'
```

Observa que esta operación es equivalente a

```
f'{3.141592:0.4f}'
```

Sin embargo, el método format tiene otros alcances. Por ejemplo.

```
[21]:  '{} {}'.format('Nombre', 'Apellido')
[21]:  'Nombre Apellido'
```

Puedes repetir una cadena asignando valores enteros consecutivos a los argumentos de `format`.

```
[22]: '{0} {0} {0} {1} {2}'.format('Vamos','sigue','adelante')
[22]: 'Vamos Vamos Vamos sigue adelante'
```

Y también puedes trabajar con variables dentro del argumento de `format`

```
[23]: '{nombre} {apellido}'.format(nombre='Rogelio',apellido='García')
[23]: 'Rogelio García'
[24]: '{apellido} {nombre}'.format(nombre='Rogelio',apellido='García')
[24]: 'García Rogelio'
```

Anteriormente utilizaste el símbolo + para concatenar cadenas. Además de esta operación puedes repetir una misma cadena, de manera similar al snippet [22]

```
[25]: s1='Hola'
[26]: s2='mundo'
[27]: s1 +=' '+s2
[28]: s1
[28]: 'Hola mundo'
```

En el snippet [27] se concatenan los arreglos `s1`, un espacio en blanco y `s2`. Además, el operador += asigna a `s1` el resultado de la concatenación.

Si requieres repetir una cadena.


```
[29]:  eco='feliz '
[30]:  eco*=4
[31]:  yo='estoy'
[32]:  estado=yo+' '+eco
[33]:  estado
[33]:  'estoy feliz feliz feliz feliz '
```

En el snippet [30] la cadena `'feliz '` se repite 4 veces y se guarda en `eco`, mientras que el snippet [32] concatena los arreglos `yo`, un espacio en blanco y `eco`.

8.3 Otras opciones para las cadenas

Existen varios métodos para quitar espacios en blanco, en todas ellas las cadenas son objetos inmutables, por lo que siempre que se realiza tal tarea, aparece una cadena nueva con los espacios en blanco eliminados.

Por ejemplo, observa el siguiente enunciado (cadena)

```
[1]: enunciado = '\t \n me gusta escuchar música alegre \t\t \n'
[2]: enunciado
[2]: '\t \n me gusta escuchar música alegre \t\t \n'
```

Si requieres eliminar todos los espacios en blanco (esto incluye tabulaciones y saltos de línea), al inicio y al final del enunciado. El método `strip` realiza esta tarea

```
[3]:   enunciado.strip()
[3]:   'me gusta escuchar música alegre'
```

Para eliminar los espacios en blanco a la izquierda (inicio) de la cadena.

```
[4]:   enunciado.lstrip()
[4]:   'me gusta escuchar música alegre \t\t \n'
```

O los espacios de la derecha (final) de la cadena.

```
[5]:   enunciado.rstrip()
[5]:   '\t \n me gusta escuchar música alegre'
```

Si eres observador, habrás notado que no es necesario volver a llamar a la variable enunciado para aplicar, en cada caso, las diferentes variantes del método `strip`. Esto ocurre porque la cadena original nunca se modifica (es inmutable).

En los capítulos anteriores viste la forma de cambiar las letras mayúsculas y minúsculas de una cadena. Python realiza esta tarea de una forma más compacta a través de los métodos `capitalize` y `title`. Como el método `strip`, en ambos casos se crea una copia de la cadena original y se obtiene una nueva.

```
[6]:  'me gusta eschuchar música alegre'.capitalize()
[6]:  'Me gusta eschuchar música alegre'
```

El método capitalize, cambia la primera letra de una cadena por una mayúscula. Si la letra ya es mayúscula, no se presenta ningún cambio.

```
[7]:  'me gusta eschuchar música alegre. bailemos'.title()
[7]:  'Me Gusta Eschuchar Música Alegre. Bailemos'
```

El método title, cambia la primera letra de cada palabra de una cadena por una mayúscula. Si la letra ya es mayúscula, en alguna de las palabras, en éstas, no se presenta ningún cambio. Además de dar formato a las cadenas, Python permite compararlas. Esto se realiza con los valores numéricos que tiene cada carácter. Aquí algunos ejemplos.

```
[8]:  'Python' == 'python'
[8]:  False
[9]:  'Python' != 'python'
[9]:  True
[10]: 'Python' > 'python'
[10]: False
[11]: 'Python' < 'python'
[11]: True
[12]: 'Python' >= 'python'
[12]: False
[13]: 'Python' <= 'python'
[13]: True
```

Para contar el número de veces que aparece un trozo de cadena (substring) en una cadena más grande, Python cuenta con el método count. Por ejemplo.

```
[14]: enunciado='quizás si quizás no quizás es una posibilidad'
[15]: enunciado.count('quizás')
[15]: 3
```

Si añades un segundo argumento, éste indica el índice de inicio de la búsqueda (recuerda que los índices inician en 0).

```
[16]: enunciado.count('quizás',8)
[16]: 2
```

El argumento 8, es el índice del caracter a partir del cual inicia la búsqueda, en el caso de la cadena

```
'quizás si quizás no quizás es una posibilidad'
```

El caracter con índice 8 es la i, antes del espacio en blanco del segundo quizás.

Si añades un segundo y tercer argumento al método count, estarás indicando el campo en donde Python debe realizar la búsqueda.


```
[17]: enunciado.count('quizás',8,18)
[17]: 1
```

Los argumentos 8, 18 indican los índices de los caracteres entre los cuales se debe realizar la búsqueda, en el código anterior, Python busca la cadena quizás entre las letras señaladas con verde.

```
'quizás si quizás no quizás es una posibilidad'
```

El método index busca una subcadena dentro de una cadena, e indica el primer índice en donde aparece. Si la subcadena no está incluida, ocurre un error del tipo ValueError.

```
[18]: enunciado.index('quizás')
[18]: 0
```

El resultado indica que la cadena 'quizás' aparece en la primera palabra de la cadena.

```
[19]: enunciado.rindex('quizás')
[19]: 20
```

El método rindex realiza la misma tarea que index, pero inicia la búsqueda desde la parte final de la cadena. En el caso de la cadena

```
'quizás si quizás no quizás es una posibilidad'
```

El caracter s, de la tercera palabra quizás, ocupa el índice 20 iniciando desde la parte final.
Si solo necesitas conocer si en la cadena se encuentra una subcadena, es posible utilizar los operadores in o not in.

```
[20]: 'Si' in enunciado
[20]: False
[21]: 'Si' in enunciado
[21]: True
[22]: 'Si' not in enunciado
[22]: True
```

Los métodos startswith y endswith regresan True si la cadena inicia o termina, respectivamente, con alguna subcadena específica.

```
[23]: enunciado.startswith('quizás')
[23]: True
[24]: enunciado.startswith('si')
[24]: False
[25]: enunciado.endswith('posibilidad')
[25]: True
[26]: enunciado.endswith('una')
[26]: False
```

Al leer un segmento de texto, automáticamente nuestra percepción visual nos hace mirarlo como un conjunto de palabras individuales, a este proceso le llamaremos tokenizar y a cada palabra la llamaremos *token*.

Para *tokenizar* una cadena puedes utilizar un delimitador personalizado como una coma o un espacio en blanco. Ve el siguiente ejemplo

```
[27]: letras='A,b,e,c,e,d,a,r,i,o'
[28]: letras.split(',')
[28]: ['A', 'b', 'e', 'c', 'e', 'd', 'a', 'r', 'i', 'o']
```

Observa que el método `split` sin argumentos tokeniza una cadena separando palabras con cada espacio en blanco o como en este caso, con una coma, al finalizarla función regresa una lista de tokens.

Si escribes un entero en la segunda parte del argumento, este, especifica el número máximo de divisiones de la cadena.

```
[29]:  letras.split(',',4)
[29]:  ['A', 'b', 'e', 'c', 'e,d,a,r,i,o']
```

El método `join` concatena las cadenas en sus argumentos el cual debe ser un iterable y contienen sólo cadenas, en otro caso ocurrirá una excepción tipo error del tipo `TypeError`.

```
[30]: unir_palb=['p','a','l','a','b','r','a']
[31]: ','.join(unir_palb)
[31]: 'p,a,l,a,b,r,a'
```

El siguiente snippet une los elementos de una comprensión de lista.

```
[32]: ','.join([str(n) for n in range(1,11)])
[32]: '1,2,3,4,5,6,7,8,9,10'
```

Si hay una gran cantidad de texto en una cadena, puede ser deseable separar la cadena en una lista de líneas. El método `splitlines` regresa una lista de cadenas que representan líneas de texto. Como en el siguiente código.

```
[33]: verso="""Ayer pensé en tí,
      creí que aquí estabas,
      y cuando mi rostro volví,
      note que me mirabas"""
[34]: verso
[34]: 'Ayer pensé en tí,\ncreí que aquí estabas,\ny cuando
      mi rostro volví,\nnote que me mirabas'
[35]: verso.splitlines()
[35]: ['Ayer pensé en tí,',
      'creí que aquí estabas,',
      'y cuando mi rostro volví,',
      'note que me mirabas']
```


Con el argumento `True` en `splitlines`, se muestran los respectivos saltos de línea.

```
[36]: verso.splitlines(True)
[36]: ['Ayer pensé en tí,\n',
      'creí que aquí estabas,\n',
      'y cuando mi rostro volví,\n',
      'note que me mirabas']
```

Cuando sea necesario, es posible verificar si un caracter es de cierto tipo. Por ejemplo el método `isdigit`.

```
[37]:  '123'.isdigit()
[37]:  True
[38]:  '3.1416'.isdigit()
[38]:  False
[39]:  '-34'.isdigit()
[39]:  False
```

El snippet [37] ha sido calificado como verdadero porque contiene caracteres que representan números dígitos, los otros dos han sido calificados como falsos, ya que contienen caracteres que no son numéricos como el punto decimal del snippet [38] y el signo del snippet [39].

Esta misma estrategia puede emplearse para identificar caracteres alfanuméricos

```
[40]:  'Code84'.isalnum()
[40]:  True
[41]:  '3.1416'.isalnum()
[41]:  False
[42]:  'Calle 5 No 314'.isalnum()
[42]:  False
```

El snippet [40] ha sido calificado como verdadero porque contiene caracteres alfabéticos y numéricos, los otros dos han sido calificados como falsos, ya que contienen caracteres que no son alfanuméricos como el punto decimal del snippet [41] y los espacios del snippet [42].

Dentro de los caracteres que no son alfanuméricos, uno de ellos que es de bastante utilidad y al que está dedicado este espacio es la diagonal invertida o *backslash*. Este carácter es utilizado, por un lado, para definir secuencias de escape como los saltos de línea (`\n`) o las tabulaciones (`\t`). Por otro lado, Microsoft Windows lo emplea para separar nombres de carpetas y las ubicaciones de los archivos.

Para indicar alguna ubicación en Windows es aceptable escribir

```
[43]: ubicación='C\\usuario\\documentos\\praticas\\mi_archivo.txt'
[44]: ubicación
[44]: 'C\\usuario\\documentos\\praticas\\mi_archivo.txt'
```

A este tipo de cadenas se les reconoce como cadenas crudas (raw strings) y se pueden declarar como en el caso de f-string, con una r.

```
[45]: ubicación=r'C\usuario\documentos\praticas\mi_archivo.txt'
[46]: ubicación
[46]: 'C\\usuario\\documentos\\praticas\\mi_archivo.txt'
```

Esta estrategia permite utilizar la diagonal invertida como un carácter alfanumérico, sin el problema de que se confunda con alguna secuencia de escape.

8.4 Expresiones Regulares

Una expresión regular es un patrón de coincidencia de texto, que tiene una sintaxis muy específica. Un ejemplo de una expresión regular son los números de un teléfono. Contienen 10 dígitos y los dos primeros indican la región del país. Así que al leer un número telefónico es posible reconocer la región a la que pertenece y si está bien escrito (dígitos completos)

Otros casos de expresiones regulares son: el código postal, el número de seguro social, el registro federal de contribuyente (RFC), la clave única de registro de población (CURP), las direcciones de correo electrónico, etc. Al verificar que todos estos objetos cumplen con el patrón adecuado, estarás dando una validación de los datos ingresados.

Además de validar información, las expresiones regulares también son aplicables para:

- La extracción de datos de texto.
- Limpiar datos.
- Transformar datos en otros formatos.

Para utilizar expresiones regulares es necesario importar el módulo re.

```
[1]:  import re
```

La función más simple para expresiones regulares es `fullmatch`. A continuación, se muestran algunos ejemplos de la forma en que se puede utilizar.

```
[2]: código = '9400354'
[3]: 'Código correcto' if re.fullmatch(código,'9400355') else
     'Código incorrecto'
[3]: 'Código incorrecto'
[4]: 'Código correcto' if re.fullmatch(código,'9400354') else
     'Código incorrecto'
[4]: 'Código correcto'
```

Esta parece ser una tarea muy obvia, pero en realidad la utilizas con bastante frecuencia, por ejemplo: al escribir tu nip en un cajero automático, o tu contraseña al encender tu computadora, o la clave para activar tu celular.

Las expresiones regulares, en general contienen símbolos especiales llamados meta caracteres como

\, @, #, $ * y signos de agrupación como [ ], { }, ( )

En consecuencia, resulta necesario utilizar los `r-strings`.

```
[5]: 'Teléfono correcto' if re.fullmatch(r'\d{10}', '5512312312')
     else 'Teléfono incorrecto'
[5]: 'Teléfono correcto'
[6]: 'Teléfono correcto' if re.fullmatch(r'\d{10}', '12312312')
     else 'Teléfono incorrecto'
[6]: 'Teléfono incorrecto'
```

Los snippets [5] y [6] validan si la cadena definida por el número telefónico, contiene 10 dígitos.

También es posible verificar si, por ejemplo, el nombre de las personas está bien escrito (letra mayúscula al inicio del nombre).

```
[7]: 'Escritura válida' if re.fullmatch('[A-Z][a-z]*','rogelio')
else 'Escritura no válida'
[7]: 'Escritura no válida'
[8]: 'Escritura válida' if re.fullmatch('[A-Z][a-z]*','Laura')
else 'Escritura no válida'
[8]: 'Escritura válida'
[9]: 'Escritura válida' if re.fullmatch('[A-Z][a-z]*','Rogelio')
else 'Escritura no válida'
[9]: 'Escritura válida'
```

Aquí, `[A-Z]` verifica que la primera letra esté escrita con mayúsculas, de manera similar, `[a-z]*` verifica si las letras, después de la primera, están escritas con minúsculas. El operador *, en `[a-z]*`, verifica si hay cero o más ocurrencias de `[a-z]`.

Al escribir `[^a-z]`, estarás verificando si algún caracter no coincide con las letras minúsculas.

```
[10]: 'Correcto' if re.fullmatch('[^a-z]','O') else 'Incorrecto'
[10]: 'Correcto
[11]: 'Correcto' if re.fullmatch('[^a-z]','w') else 'Incorrecto'
[11]: 'Incorrecto'
[12]: 'Correcto' if re.fullmatch('[^a-z]','o') else 'Incorrecto'
[12]: 'Incorrecto'
```

Si requieres verificar si al menos una letra minúscula aparece en el nombre, puedes utilizar `[a-z]+`

```
[13]: 'Correcto' if re.fullmatch('[A-Z][a-z]+','Eva') else 'Incorrecto'
[13]: 'Correcto'
[14]: 'Correcto' if re.fullmatch('[A-Z][a-z]+','E') else 'Incorrecto'
[14]: 'Incorrecto'
```

Las siguientes expresiones regulares verifican si las expresiones contienen al menos cuatro dígitos.

```
[15]: 'Correcto' if re.fullmatch(r'\d{4,}','9875') else
      'Incorrecto'
[15]: 'Correcto
[16]: 'Correcto' if re.fullmatch(r'\d{4,}','987654321')
      else 'Incorrecto'
[16]: 'Correcto'
[17]: 'Correcto' if re.fullmatch(r'\d{4,}','1.002') else
      'Incorrecto'
[17]: 'Incorrecto'
```

Ahora, las expresiones regulares siguientes, verifican si las expresiones contienen entre 8 y 10 dígitos.

```
[18]: 'Correcto' if re.fullmatch(r'\d{8,10}','12')
      else 'Incorrecto'
[18]: 'Incorrecto
[19]: 'Correcto' if re.fullmatch(r'\d{8,10}','123456789')
      else 'Incorrecto'
[19]: 'Correcto'
[20]: 'Correcto' if re.fullmatch(r'\d{8,10}','123456789012')
      else 'Incorrecto'
[20]: 'Incorrecto'
```

Otras dos funcionalidades del módulo `re`, son `sub` y `split`. Observa estos ejemplos

```
[21]: import re
[22]: re.sub(r'\n',', ','Salto 1\nSalto 2\nSalto 3')
[22]: 'Salto 1, Salto 2, Salto 3'
```

La función sub requiere de tres argumentos

- El modelo de coincidencia, en este ejemplo el salto de línea (`'\n'`)
- El texto de reemplazo, en este ejemplo, la coma (`', '`)
- La cadena donde será buscado el modelo (`'Salto 1\nSalto 2\nSalto 3'`)

Al final, obtienes una cadena nueva. También puedes declarar un número máximo de reemplazos.

```
[23]: re.sub(r'\n',', ','Salto 1\nSalto 2\nSalto 3', count=1)
[23]: 'Salto 1, Salto 2\nSalto 3'
```

La función `split` puede tokenizar una cadena utilizando una expresión regular al especificar un delimitador y regresar una cadena. Haremos una separación utilizando como delimitador, una coma, o una coma seguida por espacios en blanco.

```
[24]: re.split(r',\s*','s,  e,p,a,   r, a,   d,o')
[24]: ['s', 'e', 'p', 'a', 'r', 'a', 'd', 'o']
```


Como en el caso de sub, puedes declarar un máximo de divisiones

```
[25]: re.split(r',\s*','s,  e,p,a,   r, a,   d,o', maxsplit=2)
[25]: ['s', 'e', 'p,a,   r, a,   d,o']
```

Python tiene otras funciones de búsqueda. La función `search` busca en una cadena la primera coincidencia de una expresión regular

```
[26]: import re
[27]: busca1=re.search('texto', 'Esto es solo texto de prueba')
[28]: busca1.group() if busca1 else 'no se encontró'
[28]: 'texto'
```

En este ejemplo, la función busca la subcadena `'texto'` en la cadena `'Esto es solo texto de prueba'`. Al encontrarla, muestra la cadena. Cuando no la encuentra, muestra el mensaje `'no se encontró'`.

```
[29]: busca2=re.search('no', 'Esto es solo texto de prueba')
[30]: busca2.group() if busca2 else 'no se encontró'
[30]: 'no se encontró'
```

En este ejemplo, la función busca la subcadena `'no'` en la cadena `'Esto es solo texto de prueba'`. En este caso, no se encuentra.

Los módulos `re` tienen características opcionales para realizar búsquedas más robustas. En el siguiente caso, al módulo `re` se le agrega una bandera que ignora si la coincidencia en la cadena está escrita con mayúsculas o minúsculas.

```
[31]: busca3=re.search('GUIDO','Guido Van Rossum',flags=re.
IGNORECASE)
[32]: busca3.group() if busca3 else 'no se encontró'
[32]: 'Guido'
```

El meta caracter `^` al principio de una cadena regular, es un ancla que indica a Python buscar la expresión que coincide solo con el inicio de la cadena. Si la encuentra, la muestra

```
[33]: buscador=re.search('^Guido','Guido Van Rossum es el
creador de Python',flags=re.IGNORECASE)
[34]: buscador.group() if buscador else 'no se encontró'
[34]: 'Guido'
[35]: buscador=re.search('^Rossum','Guido Van Rossum es el
creador de Python',flags=re.IGNORECASE)
[36]: buscador.group() if buscador else 'no se encontró'
[36]: 'no se encontró'
```

De manera similar, el meta caracter `$` al final de una cadena regular, es un ancla que indica a Python buscar la expresión que coincide solo al final de la cadena. Si la encuentra, la muestra.

```
[37]: buscador=re.search('Rossum$','Guido Van Rossum es el
creador de Python',flags=re.IGNORECASE)
[38]: buscador.group() if buscador else 'no se encontró'
[38]: 'no se encontró'
[39]: buscador=re.search('Python$','Guido Van Rossum es el
creador de Python',flags=re.IGNORECASE)
[40]: buscador.group() if buscador else 'no se encontró'
[40]: 'Python'
```

Para finalizar esta sección, utilizaremos un par de funciones de `re`, `findall`, que encuentra todas las subcadenas en una cadena y `finditer` que realiza la misma tarea que `findall`. La gran diferencia es que `finditer` regresa una evaluación floja de objetos que coinciden.

```
[41]: usuario='Francisco  García,  Tel_casa:52-1234-1234,
Celular:52-4321-4321'
[42]: re.findall(r'\d{2}-\d{4}-\d{4}',usuario)
[42]: ['52-1234-1234', '52-4321-4321']
```

Con `finditer`.

```
[43]: for telefono in re.finditer(r'\d{2}-\d{4}-\d{4}', usuario):
      print(telefono.group())
[43]: 52-1234-1234
      52-4321-4321
```

Naturalmente, cuando hay una gran cantidad de coincidencias, la lectura es más sencilla con `finditer` y con la bondad de no ocupar tanto espacio en la memoria de la computadora.

8.5 Pandas y Expresiones Regulares

En esta sección utilizaremos las expresiones regulares para validar la escritura del RFC.

En México, el RFC es el Registro Federal de Contribuyentes, que es una clave única alfanumérica con la que el gobierno de México, identifica a personas físicas (personas que reciben un salario) y personas morales (empresas) que lleven a cabo una actividad económica en el país.

Para una persona física, el RFC sin homoclave (sin registro oficial), consta de 4 letras y 6 dígitos. Las letras están vinculadas al nombre de la persona y los dígitos con la fecha de nacimiento.

El siguiente código crea una `Series` de Pandas con los RFC de dos usuarios. Intencionalmente uno de ellos tiene un error. El objetivo es ver la forma en que Python lo identifica este error.

```
[1]: import pandas as pd
[2]: rfc=pd.Series({'Usuario1': 'COG891201','Usuario2':
     'GUHA911020'})
[3]: rfc
```



```
[3]: Usuario1     COG891201
     Usuario2    GUHA911020
     dtype: object
```

Para definir la expresión regular y determinar las coincidencias.

```
[4]: rfc.str.match(r'\w{4}\d{6}')
[4]: Usuario1    False
     Usuario2    True
     dtype: bool
```

Como era de esperar, el Usuario1 no tiene registrado de manera correcta su RFC.

El método match aplica la expresión regular `\w{4}\d{6}` a cada elemento de `Series`, validando si los elementos tienen exactamente 4 letras y 6 dígitos. Observa que no es necesario escribir un ciclo para revisar cada uno de los elementos, match lo realiza de manera automática.

Ahora crearemos una cadena que contiene los nombres de tres países, su código ISO 3 y el prefijo telefónico.

```
[5]:  códigost=pd.Series(['México,  MEX  52','Colombia,  COL
57','Chile, CHL 56'])
[6]: códigost
[6]: 0       México, MEX 52
     1     Colombia, COL 57
     2        Chile, CHL 56
     dtype: object
[7]: códigost.str.contains(r' [A-Z]{3}')
[7]: 0    True
     1    True
     2    True
     dtype: bool
[8]: códigost.str.match(r' [A-Z]{3}')
[8]: 0    False
     1    False
     2    False
     dtype: bool
```

El snippet [7] utiliza la función `contains` para mostrar que los tres elementos de Series contienen subcadenas que coinciden con `[A-Z]{3}`. Es decir, la cadena tiene 3 letras mayúsculas. El snippet [8] emplea `match` para mostrar si alguno de los elementos en Series coincide completamente con `[A-Z]{3}`. Aunque las letras mayúsculas si aparecen, también aparecen espacios en blanco y números, por lo que no hay, en ningún caso una coincidencia completa.

CAPÍTULO 9

Archivos

9.1 Introducción

Hasta hoy, los programas o códigos que elaboraste conservan solo temporalmente los datos que haz utilizado. Sin embargo, puede resultar necesario conservar esta información en archivos con el fin de realizar consultas o hacer informes posteriores. En este capítulo aprenderás a guardar tus resultados en diferentes formatos como: TXT (texto plano), JSON (JavaScript Object Notation), y CSV (coma-separated values). Además de aprender a lidiar con algunos tipos de errores que ocurren con frecuencia al momento de programar.

Mensaje

9.2 Archivos TXT

Cuando Python lee un archivo TXT, como una secuencia de caracteres. Y, como en una lista con *n* elementos, la primera posición de la lista binaria ocupa el lugar 0 y en la última posición el lugar *n*-1.

Cada sistema operativo tiene su propio instrumento para indicar cuando termina el archivo. A este elemento se le llama `end-on-file marker`.

Por cada archivo que se abre, Python genera un archivo tipo objeto que le permite interactuar con el archivo original.

El siguiente código muestra la forma de crear un archivo de tipo TXT.

```
[1]: with open('seguidores.txt',mode='w') as seguidores:
        seguidores.write('@charlidamelio D'Amelio 108.1\n')
        seguidores.write('@addisonre Addison 76.4\n')
        seguidores.write('@zachking Zach 56.6\n')
        seguidores.write('@bellapoarch Bella 56.3\n')
```

Este archivo contiene el registro de los 4 usuarios de TikTok con más seguidores en el mundo. El código registra un handle que los identifica, el nombre y la cantidad de seguidores (en millones).

Fuente: https://marketing4ecommerce.mx/top-los-influencers-mas-seguidos-en-redes-sociales-en-el-mundo/. Última revisión 26/02/2021.

Notas

La sentencia with de Python

- Gestiona un recurso (un archivo tipo objeto para seguidores.txt) y asigna este objeto a una variable (seguidores).
- Otorga permisos a la aplicación para manipular el recurso a través de la variable.
- Llama al instrumento end-on file marker, para liberar el recurso cuando el programa finaliza.

La función `open` abre el archivo seguidores.txt asociado con el archivo tipo objeto. La función `open` tiene dos argumentos: el primero es el nombre del archivo, el segundo es el modo, que puede ser de lectura o escritura. En este caso `w`. Este modo indica que el archivo se abre para escribir (write). El archivo se guarda por default, en la misma carpeta en donde se escribe el código con extensión TXT.

La sentencia `with` asigna el objeto generado por open a la variable seguidores, por medio de `as`. Dentro de la sentencia `with`, la variable interactúa con el archivo con el método `write`, que se presenta 4 veces en el código para anexar los datos de los usuarios.

El archivo `seguidores.txt`, contiene la siguiente información

```
@charlidamelio D'Amelio 108.1
@addisonre Addison 76.4
@zachking Zach 56.6
@bellapoarch Bella 56.3
```

La tarea de revisar los contenidos a través de Python se puede realizar con el siguiente código.

```
[2]: with open('seguidores.txt', mode = 'r') as seguidores:
        print(f'{"Handle" :<20}{"Nombre" :<10}{"num_seg" :>10}')
        for record in seguidores:
            handle,nombre,num_seg = record.split()
            print(f'{handle:<20}{nombre:<10}{num_seg:>10}')

     Número              Nombre        Puntos
     @charlidamelio      D'Amelio       108.1
     @addisonre          Addison         76.4
     @zachking           Zach            56.6
     @bellapoarch        Bella           56.3
```

El archivo únicamente se abrió para revisar los contenidos, de manera que el segundo argumento de la función open tiene el modo lectura (`mode = 'r'`).

Para cambiar alguno de los registros, no es recomendable modificar directamente el archivo de texto, pues se corre el riesgo de cambiar las características del archivo, como la cantidad de caracteres o el tamaño de los campos que Python lee a través de sus rutinas.

Hacerlo a través de Python puede resultar un tanto complejo. La siguiente rutina muestra todos los pasos que se realizan.

```
[3]: seguidores=open('seguidores.txt','r')
[4]: temporal=open('temporal.txt','w')
[5]: with seguidores,temporal:
        for record in seguidores:
            handle,nombre,num_seg = record.split()
            if handle!='@charlidamelio':
               temporal.write(record)
            else:
               nuevo_nombre=' '.join([handle, 'Charlie', num_seg])
               temporal.write(nuevo_nombre +'\n')
```

Observa que cambiar un solo registro requiere:

- Copiar el registro que se quiere modificar en un archivo temporal
- Escribir la modificación que ha de realizarse
- Copiar la actualización en el archivo temporal
- Renombrar al archivo temporal con el nombre original

La desventaja de esto, es que se genera un archivo nuevo que ocupa espacio en la memoria. La ventaja es que la información original no se modifica. Aunque esta rutina puede resultar incómoda, tiene un mejor rendimiento cuando el archivo tiene una gran cantidad de registros y se requieren hacer múltiples modificaciones.

Los datos del archivo temporal de `seguidores` son:

```
@charlidamelio Charlie 108.1
@addisonre Addison 76.4
@zachking Zach 56.6
@bellapoarch Bella 56.3
```

9.3 Archivos JSON

Los archivos JSON (JavaScript Object Notation) son archivos de texto con un formato ligero fácil de leer y de escribir para los programadores y fácil de interpretar por Python. Este tipo de archivos describen objetos que utilizan algunas compañías para interactuar con algunos servicios en la nube.

Los objetos tipo JSON son similares a los diccionarios de Python. Cada objeto JSON contiene un lista con elementos agrupados con llaves, { }. Los elementos son nombres clave y sus valores separados por comas.

Por ejemplo: `{'handle':@charlidamelio, 'name':'Charlie', 'puntos': 108.1}`

Aunque también, JSON soporta arreglos con elementos separados por comas agrupados entre corchetes.

Por ejemplo: `[108.1, 76.4, 56.6 56.3]`

Pyhon tiene en sus librerías el módulo `json` que permite interactuar objetos JSON con archivos de texto.

Consideremos el siguiente diccionario referente al número de seguidores que tienen los usuarios de la aplicación TikTok.

```
[1]: seguidores_dicc={'seguidores\n':[
 {'handle':'@charlidamelio', 'name':'Charlie', 'num_seg':108.1},
 {'handle':'@addisonre', 'name':'Addison', 'num_seg':76.4},
 {'handle':'@zachking', 'name':'Zach', 'num_seg':56.6},
 {'handle':'@bellapoarch', 'name':'Bella', 'num_seg':56.3}]}
```

Para escribir este objeto con el formato JSON.

```
[2]: import json
[3]: with open('seguidores.json','w') as seguidores:
        json.dump(seguidores_dicc,seguidores)
```

El snippet [6] abre archivo `seguidores.json` y utiliza la función `dump` del módulo json para serializar al diccionario seguidores_dicc dentro del archivo. El archivo `seguidores.json` contiene a los elementos. Como en los archivos TXT, la función open tiene en el segundo argumento la opción `'w'` para escribir.

Para recuperar los datos del archivo JSON es posible utilizar la función `load`. La función load del módulo json realiza una lectura de todos los elementos del archivo JSON y convierte a este elemento en un objeto de Python, a esto se le llama deserializar los datos.

```
[4]: with open('seguidores.json','r') as seguidores:
       seguidores_json=json.load(seguidores)
[5]: seguidores_json
```



```
[5]: {'seguidores':
 [{'handle': '@charlidamelio', 'name': 'Charlie', 'num_seg': 108.1},
 {'handle': '@addisonre', 'name': 'Addison', 'num_seg': 76.4},
 {'handle': '@zachking', 'name': 'Zach', 'num_seg': 56.6},
 {'handle': '@bellapoarch', 'name': 'Bella', 'num_seg': 56.3}]}
```

Como en los archivos TXT, la función open ahora tiene en el segundo argumento la opción `'r'` de lectura.

Justo como esperabas, ya puedes acceder a los contenidos del diccionario. Para obtener la lista del diccionario asociada a la llave `'seguidores'`:

```
[6]: seguidores_json['seguidores']
[6]: [{'handle': '@charlidamelio', 'name': 'Charlie', 'num_seg': 108.1},
{'handle': '@addisonre', 'name': 'Addison', 'num_seg': 76.4},
{'handle': '@zachking', 'name': 'Zach', 'num_seg': 56.6},
{'handle': '@bellapoarch', 'name': 'Bella', 'num_seg': 56.3}]}
```

Y para obtener los registros individuales

```
[7]: seguidores_json['seguidores'][0]
[7]: {'handle': '@charlidamelio', 'name': 'Charlie', 'num_seg': 108.1}
[8]: seguidores_json['seguidores'][3]
[8]: {'handle': '@bellapoarch', 'name': 'Bella', 'num_seg': 56.3}
```

Nota **PREVENTIVA**.

Otra forma de serializar y deserializar es con el módulo `pickle` de Python. Sin embargo, la documentación de Python hace algunas advertencias acerca del módulo `pickle`:

- Los archivos pickle pueden ser hackeados fácilmente, por lo que no es nada confiable abrir archivos con este formato.
- El protocolo de pickle puede llevar a una serialización profundamente compleja de objetos de Python que pueden derivar en la interacción no autorizada con otras aplicaciones, particularmente si los datos fueron creados con objetivos fraudulentos de un experto.

Cuando abres un archivo, además de los modos de lectura (r) y de escritura (w), Python cuenta con otros modos de trabajar archivos.

Fase	Tarea
'r'	Abre un archivo para lectura. Esta es la opción por default.
'r+'	Abre un archivo para lectura y escritura.
'a'	Abre un archivo que se agrega al final. Crea al archivo si no existe. Los datos nuevos son agregados en la parte final del archivo.
'a+'	Abre un archivo para lectura que se agrega al final. Crea al archivo si no existe. Los datos nuevos son agregados en la parte final del archivo.
'w'	Abre un archivo para escritura. Los contenidos del archivo existente son borrados.
'w+'	Abre un archivo para lectura y escritura. Los contenidos del archivo existente son borrados.

9.4 Manipulación de Excepciones

Existen varios tipos de excepciones que pueden ocurrir cuando trabajas con archivos, por ejemplo:

- Excepciones del tipo FileFoundError, ocurren cuando intentas abrir un archivo que no se encuentra
- Excepciones del tipo PermissionsError, suceden si, por ejemplo, quieres guardar un archivo en una carpeta en la cual no tienes permiso para hacerlo.
- Excepciones del tipo ValueError, con mensaje de error 'I/O/operation on closed file' que pasan cuando intentas en escribir en un archivo que ya ha sido cerrado.
-

Y otras excepciones que ya se han presentado como ZeroDivisionError cuando se presenta una división entre cero.

```
[1]:  4/0
```

```
[1]:  ---------------------------------------------------
ZeroDivisionError         Traceback (most recent call last)
<ipython-input-14-221068dc2815> in <module>
----> 1 4/0

ZeroDivisionError: division by zero
```

Y la excepción del tipo `ValueError`, que ocurre, por ejemplo, cuando tratas de convertir un arreglo en un entero.

```
[2]:  int('hola mundo')
```

```
[2]:  ---------------------------------------------------
ValueError                Traceback (most recent call last)
<ipython-input-15-ccd7a6b7e0ea> in <module>
----> 1 int('hola mundo')

ValueError: invalid literal for int() with base 10: 'hola mundo'
```

En ambos casos la ejecución del programa se detiene para mostrar el mensaje de error. Para dar continuidad a la ejecución del programa, si es que hay más tareas por realizar. La estrategia es reconocer cuando ocurre la excepción y manipularla para que el programa siga con la totalidad de las tareas. Aquí está un ejemplo que evita la excepción ZeroDivisionError.

Ejemplo 1

```
# programa1_C9.py
"""Manipulación de la excepción ZeroDivisionError"""

while True:
#diferentes casos de división
    try:
        numerador=int(input('Ingresa el numerador'))
        denominador=int(input('Ingresa el denominador'))
        cociente=numerador/denominador
    except ValueError:
        print('Debes ingresar dos números enteros\n')
    except ZeroDivisionError:
        print('Hay una división entre cero\n')
    else:
        print(f'{numerador:.1f}/{denominador:.1f}={cocien-
te:.1f}')
        break
```

Algunos resultados son:

```
Ingresa el numerador 26
Ingresa el denominador 12.3
Debes ingresar dos números enteros

Ingresa el numerador 132
Ingresa el denominador 0
Hay una división entre cero

Ingresa el numerador 36
Ingresa el denominador 9
36.0/9.0=4.0
```

Esta estrategia permite seguir trabajando sin que el programa se detenga.

En el ejemplo anterior, Python utiliza la sentencia `try`. La condición `try` es seguida de una o más excepciones, cada `except` especifica un tipo de excepción a manipular. Después del último `except`, una condición opcional else, se ejecuta, si ninguna excepción ocurre.

El programa del ejemplo 1, trata de dividir dos números enteros, por lo que pide al usuario introducir dos números: numerador y denominador. Si alguno de los dos números no es entero, se comete una excepción tipo `ValueError`, y solicita al usuario ingresar nuevamente los valores. Si se asigna al denominador el valor de cero, ocurre una excepción del tipo `ZeroDivisionError`, por lo que vuelve a solicitar al usuario introducir nuevos valores. Cuando se introducen los dos valores son enteros y el denominador no es cero, el programa muestra la división de los dos valores.

9.5 Archivos CSV

Este es uno de los tópicos más populares en Python, pues representa una herramienta poderosa que se utiliza en Ciencia de Datos. En esta sección procesarás archivos CSV con pandas de Python.

Para iniciar, crearemos un archivo de seguidores con los registros de los usuarios de TikTok que ya haz utilizado.

```
[1]: import csv
[2]: with open('seguidores.csv', mode='w',newline='w') as seguidores:
         writer=csv.writer(seguidores)
         writer.writerow(['@charlidamelio', 'Charlie', 108.1])
         writer.writerow(['@addisonre', 'Addison', 76.4])
         writer.writerow(['@zachking', 'Zach', 56.6])
         writer.writerow(['@bellapoarch', 'Bella', 56.3])
```

La extensión en el archivo .csv, indica que al archivo tendrá un formato de archivo CSV. La función `writer` del módulo `csv` regresa un objeto que escribe datos CSV en el archivo objeto especificado. Cada llamada del método `writerow` recibe un iterable que almacena en el archivo. Este método utiliza listas para cada registro. Por default `writerow` delimita los valores con comas, peo puedes utilizar un delimitador personalizado.

Los elementos en el archivo son:

Los datos del archivo temporal de `seguidores` son

```
@charlidamelio,Charlie,108.1
@addisonre Addison,76.4
@zachking,Zach,56.6
@bellapoarch,Bella,56.3
```

Los archivos CSV no contienen espacios en blanco después de las comas, aunque algunas veces este detalle mejora la lectura de los archivos.

Para leer los datos del archivo, puedes utilizar el siguiente código

```
[3]: with open('seguidores.csv','r', newline='') as seguidores:
        print(f'{"Handle":<20}{"Nombre":<10}{"Seguidores":>10}')
        lectura=csv.reader(seguidores)
        for record in lectura:
            handle, nombre, num_seg=record
            print(f'{handle:<20}{nombre:<10}{num_seg:>10}')

Handle              Nombre    Seguidores
@charlidamelio      Charlie        108.1
@addisonre          Addison         76.4
@zachking           Zach            56.6
@bellapoarch        Bella           56.3
```

La función `reader` del módulo `csv` regresa un objeto que lee los datos del formato CSV del archivo objeto.

Si algún campo dentro del archivo necesita alguna coma, es posible hacer una modificación sencilla, por ejemplo:

```
writer.writerow(['@charlidamelio', "D'Amelio, Charlie", 108.1])
```

Con esta modificación Python interpreta a `"D'Amelio", "Charlie"` como un solo elemento, además de que permite anexar el apostrofe en D'Amelio.

El detalle de la coma es importante, pues si escribes `"D'Amelio", "Charlie"`, estás indicando a Python que el registro tiene 2 elementos y no uno solo. Esto provocará un error, ya que en el ciclo for para leer el archivo, la lista solo tiene tres elementos para cada registro: handle, nombre y número de seguidores.

9.6 Lectura de archivos CSV con DataFrame de Pandas

A las personas que se inscriben en un gimnasio se les toman ciertos datos, como el género, el peso (en kg), el diámetro de la cintura (en cm), el número de pulsaciones por minuto. Y se les pide que realicen una prueba de acondicionamiento físico en la que realizan lagartijas, medias sentadillas y saltos, hasta que el usuario decide detenerse.

El archivo tiene nombre `datos_gimnasio.csv`, y está guardado en la carpeta donde se escriben los códigos de Python, con el objetivo de facilitar su llamado.

Para que Python lea el archivo csv.

```
[1]:  import pandas as pd
[2]:  datosgym=pd.read_csv('datos_gimnasio.csv')
```

Este conjunto de datos contiene los registros de 20 usuarios con 7 variables.

```
[3]: pd.set_option('precision',1) # muestra solo un valor decimal
[4]: datosgym.head()
[4]: codigo_user  Género  Peso  Cintura  Pulso  Lagartijas  Sentadilla Saltos
0    usuario1          M  86.6     91.4     50           5         162     60
1    usuario2          M  85.7     94.0     52           2         110     60
2    usuario3          M  87.5     96.5     58          12         101    101
3    usuario4          F  73.5     88.9     62          12         105     37
4    usuario5          M  85.7     88.9     46          13         155     58
```

El método `head` de DataFrame muestra por default las primeras 5 filas del conjunto de datos. Cuando se solicita mostrar los datos sin esta opción, y la base de datos es muy grande, Python muestra las primeras 30 filas y primeras 30 columnas.

Para mostrar las últimas 5 filas utiliza el método `tail` de DataFrame.

```
[5]: datosgym.tail()
[5]: codigo_user  Género  Peso  Cintura  Pulso  Lagartijas  Sentadilla Saltos
15    usuario16        M  91.6     94.0     62          12         210    120
16    usuario17        F  79.8     94.0     54           4          60     25
17    usuario18        M  71.2     81.3     52          11         230     80
18    usuario19        F  70.8     83.8     54          15         225     73
19    usuario20        F  62.6     83.8     68           2         110     43
```

Para modificar los encabezados del DataFrame

```
[6]: datosgym.columns=['Código','Género','Peso','Cintura','Pulso',
     'Lagartijas','Sentadillas','Saltos']
[7]: datosgym.head()
[7]: Código   Género  Peso  Cintura  Pulso  Lagartijas  Sentadillas Saltos
0  usuario1       M  86.6     91.4     50           5          162     60
1  usuario2       M  85.7     94.0     52           2          110     60
2  usuario3       M  87.5     96.5     58          12          101    101
3  usuario4       F  73.5     88.9     62          12          105     37
4  usuario5       M  85.7     88.9     46          13          155     58
```

Y con la función `describe` de Pandas, es posible obtener un resumen de las estadísticas descriptivas del conjunto de datos.

```
[8]: datosgym.describe()
[8]:    Peso  Cintura   Pulso  Lagartijas  Sentadillas  Saltos
count   20.0     20.0    20.0        20.0         20.0    20.0
mean    81.0     89.9    56.1         9.4        145.6    70.3
std     11.2      8.1     7.2         5.3         62.6    51.3
min     62.6     78.7    46.0         1.0         50.0    25.0
25%     72.9     83.8    51.5         4.8        101.0    39.5
50%     79.8     88.9    55.0        11.5        122.5    54.0
75%     86.9     94.0    60.5        13.2        210.0    85.2
max    112.0    116.8    74.0        17.0        251.0   250.0
```


Para revisar las estadísticas de las mujeres (género F)

```
[9]: (datosgym.Género == 'F').describe()
[9]: count       20
     unique       2
     top       True
     freq        11
     Name: Género, dtype: object
```

Donde

- `count`, es el total de elementos en el conjunto de datos
- `unique`, es el número de valores únicos (hay 2, F y M)
- `top`, es el valor más frecuente que ocurre en los resultados
- `freq`, es el número de ocurrencias del valor top.

La visualización es útil para entender el comportamiento de los datos. Pandas puede interactuar con `Matplotlib` para elaborar gráficos descriptivos. Para hacer un diagrama, primero debes habilitar `Matplotlib` de Python.

```
[10]: import matplotlib.pyplot as plt
[11]: %matplotlib
[12]: histogram=datosgym.hist()
```

En este caso, hist crea un histograma para cada columna numérica.

C A P Í T U L O 1 0

Programación orientada a objetos

10.1 Introducción

La programación orientada a objetos la puedes entender como una estrategia o como un modelo que permite el diseño y el desarrollo de programas (software).

Con este capítulo inicia una serie de tópicos especializados en programación. La continuidad en los contenidos busca, con ejemplos sencillos, adentrarte en la programación orientada a objetos, esperando que este postre, que saborean quienes programan, tenga un sabor dulce y profundo, sin llegar a fastidiar al paladar.

Toma asiento, acerca tus instrumentos y prueba con calma este pequeño postre de programación.

10.2 Métodos y Clases

Por objeto debes entender que es cualquier cosa que puedes percibir, y que con tu experiencia conoces y puedes describir sus características.

Por ejemplo, si te gustan las mascotas, un perro puede ser un objeto. Pero el perro que tienes en casa tiene cierta raza: un pug, un schnauzer o un pitbull. Y estos perros pueden cumplir tareas como la de ser un animal de compañía o de seguridad en casa. A la imagen común de mascota, le llamaremos plantilla o molde. Y partiendo de esta plantilla es posible visualizar otras versiones de una mascota, como: perro, gato, pájaro, etc, que tienen características diferentes. A éstas les llamaremos atributos.

En términos de programación, se pueden elaborar programas que caractericen a un objeto. Estos programas pueden servir de plantillas, de las que puedes utilizar el código que necesites para elaborar programas que resuelvan problemas más complejos.

Con esta perspectiva de un objeto, una clase o método, es como una especie de anaquel con diferentes herramientas, que puedes aplicar a diferentes objetos.

El trabajo inicia diseñando algunas herramientas que pondrás en tu anaquel personal. Luego la tarea será, utilizar tres herramientas del vasto almacén de Python.

El siguiente código crea tu primera clase personalizada.

```
[1]:class Mascota:
        #atributos
        tipo='pequeño'
        raza='chihuahua'
        tarea='guardián'
[2]: perro=Mascota() #Objeto
[3]: """ Atributos de objeto."""
[4]: print(perro.raza)
[4]: chihuahua
[5]: print(perro.tarea)
[5]: guardián
```

Una convención en Python al definir los nombres de las clases, es que el nombre de la clase debe empezar con una letra mayúscula.

Dentro de la clase están definidos los atributos, en este caso, tipo, raza y tarea.

El objeto, en este ejemplo puede ser un gato, o un ave. Aquí se definió como un perro.

Las últimas líneas, son para mostrar un par de atributos del objeto perro. Para que la clase quede bien definida, no son necesarias. Sin embargo, se añaden de prueba para entender la forma de invocar a los atributos.

Si es necesario cambiar alguno de los atributos, puedes realizarlo con la función setattr, de la siguiente manera.

```
setattr(perro, 'tarea', 'amigo')
print(perro.raza)
print(perro.tarea)
chihuahua
amigo
```

Para borrar alguno de los atributos que estás utilizando, la función delattr permite realizar esta tarea

```
delattr(perro,'tarea')
print(perro.tarea)
```



```
-------------------------------------------------------
AttributeError       Traceback (most recent call last)
<ipython-input-9-4747889cfa6f> in <module>
----> 1 print(perro.tarea)
```

```
AttributeError: 'Mascota' object has no attribute 'tarea'
```

Como en el caso de las sentencias for e if, los atributos se definen dentro de una identación (sangría) al escribirlo fuera, no reconocerá este atributo.

Supongamos que requieres hacer el registro, con información básica de tres aspirantes Eva, Laura y Luis a un puesto de diseñador web. Esta información básica es: edad, experiencia laboral y estado civil.

En este caso tiene varios objetos, los cuales son Eva, Laura y Luis. La clase la podemos nombrar, por ejemplo, registro. Los atributos son entonces

- Edad
- Experiencia laboral
- Estado civil

Una forma alternativa de definir objetos y atributos, es la siguiente

```
class Registro:
    pass
```

Hasta aquí se crea la clase Registro, ahora definiremos los atributos para cada objeto. La línea anterior, pass hace que Python salga del ambiente clase para ser llamada posteriormente. Ahora se crean objetos en la clase Registro.

```
eva=Registro()
laura=Registro()
luis=Registro()
eva.edad = 30
eva.experiencia = 6
eva.edo_civ = 'soltera'
laura.edad = 32
laura.experiencia = 5
laura.edo_civ = 'casada'
luis.edad = 28
luis.experiencia = 1
luis.edo_civ = 'soltero'
```

Hasta aquí está definida la clase, y los objetos con sus atributos. Las líneas siguientes permiten ver los atributos

```
print(luis.experiencia)
print(eva.edo_civ)
print(laura.edad)
```

Los valores de los atributos para cada objeto tienen la siguiente sintaxis.

```
laura.edo_civ = 'casada'
```

Objeto Atributo Valor

Otras de las características de los objetos, además de valores. Son las acciones que puede realizar, a estas acciones les llamaremos métodos. Un método es una función que realizará el objeto y que está definida dentro de una clase. El siguiente ejemplo es una muestra sencilla de las acciones que se pueden realizar.

```
class Oferta:
    def descuento(self):
        self.costo=1200
        self.desc=0.15
```

En este caso, la clase se llama Oferta, y el método descuento. Este programa tiene la tarea de calcular la cantidad que será descontada del costo. self es un parámetro de todos los métodos. Este parámetro es un acuerdo para usuarios de Python y se utiliza para acceder a los atributos del objeto. Es importante aclarar que existen métodos que tienen otros parámetros, además de self. Hasta aquí se ha creado la clase y el método. Ahora construiremos un objeto llamado rebaja.

```
rebaja=Oferta()
rebaja.descuento()
```

Esta línea indica que el objeto está invocando al método. El único que hay en este caso es descuento.

Las dos líneas anteriores indican que el objeto esta en la clase Oferta y va a utilizar el método descuento. Hasta aquí ya tienes la clase, el método y el objeto. Sin embargo, en el código no se ha calculado la rebaja. Para mostrar la rebaja puedes usar nuevamente la función print.

```
print(rebaja.costo*rebaja.desc)
180
```

Esta forma de definir al método tiene la sintaxis.

```
class Nombre_clase:
     def nombre_metodo(self):
         self.nombre_variable=valor_o_tarea
```

Tal parece que es demasiado código solo para realizar la multiplicación, pero tiene grandes beneficios en problemas más complejos.

Otra forma alternativa de construir métodos es a través de la función __init__.

10.3 La clase Cuenta

Observa este ejemplo:

```
from decimal import Decimal
class Cuenta:
    def __init__(self,nombre,ahorro):
  if ahorro<Decimal('0.00'):
raise ValueError('El capital inicial debe ser >=$0.00')
self.nombre=nombre
self.ahorro=ahorro
def deposito(self,cantidad):
if cantidad<Decimal('0.00'):
raise ValueError('La cantidad debe ser positiva')
self.ahorro+=cantidad
```

Como antes, hasta aquí se ha creado la clase, y dos métodos, pero no se ha definido ningún objeto. Si ejecutas estas líneas observarás que, aunque no muestra error, no retorna nada, porque la clase no se ha aplicado a ningún objeto.

El método __init__ cambia la forma de inicializar el método. Cuando la clase se aplica a un objeto, es lo que primero se ejecuta.

El primer método permite generar un objeto, una cuenta de ahorro designado a nombre y con un capital inicial ahorro. Esta rutina permite validar la cantidad que se asigna al ahorro, verificando que la cantidad sea mayor o igual a 0. Si ingresas un valor negativo, muestra una excepción del tipo ValueError.

El segundo método permite realizar un depósito con un valor definido por cantidad. Este valor se acumula en el capital inicial. Aquí también se valida cantidad. Si cantidad es un valor negativo ocurre un error del tipo ValueError.

Aplicaremos esta clase a un caso particular con el fin de observar el comportamiento de la rutina.

```
cuenta1=Cuenta('Juan',2000)
```

En este caso se define el objeto cuenta1 en la clase Account, con parámetros Juan y 2000. Juan es quien abre la cuenta de ahorro con un capital inicial de $2000. Para verificar:

```
print(cuenta1.nombre)
Juan
print(cuenta1.ahorro)
2000
```

Si Juan hace un depósito de $2000, entonces, el programa debe acumular esta cantidad en el ahorro

```
cuenta1.deposito(500)
print(cuenta1.ahorro)
2500
```

Al tratar de ingresar en ahorro una cantidad negativa

```
cuenta1.deposito(-1000)
```

```
---------------------------------------------------------
ValueError                Traceback (most recent call last)
<ipython-input-16-4386102c7bc1> in <module>
----> 1 cuenta1.deposito(-1000)

<ipython-input-10-ec27d46eea27> in deposito(self, cantidad)
8   def deposito(self,cantidad):
9       if cantidad<Decimal('0.00'):
---> 10   raise ValueError('La cantidad debe ser positiva')
11      self.ahorro+=cantidad

ValueError: La cantidad debe ser positiva
```

10.4 Herencia

En los anteriores ejemplos siempre estuvo presente una sola clase. Ahora revisarás el caso en que se encuentran definidas varias clases, pero con la característica, de que hay una clase inicial que las demás invocan. A esto se le llama herencia, y constituye una de las características fundamentales en la programación orientada a objetos. A esta clase inicial se le llama padre, y a las otras se les llama clases hijos, de aquí el nombre de herencia.

10.5 La clase Personal

Esta clase ha sido nombrada así, pensando en las características que tiene el personal que labora en alguna fábrica o institución. El siguiente código crea la clase Personal

```
class Personal:
    pass
   def __init__(self,nombre,profesión,cargo):
       self.nombre=nombre
       self.profesión=profesión
       self.cargo=cargo
   def detalle(self):
      return '{} es un {} y tiene el cargo de {}'.format(self.nombre,
self.profesión, self.cargo)
```

El constructor __init__ indica que los objetos para la clase Personal deben tener 3 parámetros, nombre, profesión y cargo. Al llamar a algún objeto en esta clase, regresa el método llamado detalle, que muestra la información básica que tiene algún elemento del personal.

Ahora se muestran otras dos clases con la peculiaridad de que ambas llaman a la estructura de la clase Personal. Al hacer esto, tales clases heredan la estructura de la primera.

```
class pers_nuevo(Personal):
    def contratacion(self,contrato):
      return '{} tiene contrato: {}'.format(self.nombre,contrato)
class pers_advo(Personal):
    def contratacion(self,contrato):
      return '{} tiene contrato: {}'.format(self.nombre,contrato)
```

Ahora utilizaremos el constructor para diseñar un objeto de la clase pers_nvo que tiene los atributos de Personal y el método detalle.

```
ingeniero=pers_nuevo('Paco','Ingeniero en Sistemas',"Supervisión")
print(ingeniero.detalle())
Paco es un Ingeniero en Sistemas y tiene el cargo de Supervisión
```

También puedes solicitar del objeto ingeniero, con los atributos de las otras clases

```
print(ingeniero.contratacion('Temporal'))
Paco tiene contrato: Temporal
```

Nuevamente la clase Cuenta

Utilizaremos nuevamente esta clase, pero definiremos otras clases que realizan otras tareas. En esta primera parte se encuentran separadas dos operaciones que se pueden realizar en una cuenta bancaria, un depósito y un retiro, por lo que el programa original tiene algunos cambios

```
from decimal import Decimal
class Cuenta:
def __init__(self,nombre,capital): if capital<Decimal('0.00'):
raise ValueError('El capital inicial debe ser >=$0.00') self.nombre=nombre
self.capital=capital
def informacion(self):
return '{} tiene en su cuenta ${}'.format(self.nombre,self.capital)
```

Prácticamente lo único que cambió fue el objeto de ahorro, por el objeto capital.

Luego, se definen dos subclases o clases hijo, de la clase Cuenta.

```
class Ahorro(Cuenta):
    def cantidad(self,deposito):
        if deposito<Decimal('0.00'):
            raise ValueError('La cantidad a depositar debe
ser positiva')
```

```
        self.capital+=deposito
        return '{} tiene ahora en su cuenta {}'.format(self.
nombre,self.capital)
```

Esta clase define la clase ahorro. Para prevenir errores, tiene una excepción cuando deposito es un valor negativo. Si deposito no cumple la excepción, el valor deposito se suma al valor de capital. Observa que el argumento de la clase ahorro es la clase padre Cuenta.

```
class Gasto(Cuenta):
    def cantidad(self,retiro):
        if retiro<Decimal('0.00'):
             raise ValueError('La cantidad a retirar debe
             ser positiva')
        #if retiro>capital:
        #   raise ValueError('No tiene fondos suficientes')
        self.capital-=retiro
        return '{} tiene ahora en su cuenta {}'.format(self.
        nombre, self.capital)
```

Esta clase define la clase Gasto. Para prevenir errores, tiene una excepción cuando retiro es un valor negativo. Cuando retiro no cumple la excepción, el valor retiro se resta al valor de capital. Como el argumento de la clase Gasto es Cuenta. Gasto es también una clase hijo de Cuenta.

Hasta aquí están definidas las clases. Si ejecutas el programa observarás que no regresa nada. Esto se debe a que no está aún definido ningún objeto. En las dos clases se llamó al método cantidad, esto es posible ya que la tarea que realizará el método depende del lugar donde se defina el objeto.

Ahora se definen dos objetos cuenta1 en Ahorro y cuenta2 en Gasto.

```
cuenta1=Ahorro('Juan',2000)
print(cuenta1.informacion())
Juan tiene en su cuenta $2000 print(cuenta1.capital)
2000
cuenta1.cantidad(1500)
'Juan tiene ahora en su cuenta 3500'
cuenta2=Gasto('Paco',5000)
print(cuenta2.informacion())
Paco tiene en su cuenta $5000
cuenta2.cantidad(50)
'Paco tiene ahora en su cuenta 4950'
print(cuenta2.capital)
4950
```

Ahora el usuario retira 5000

```
cuenta2.cantidad(5000)
'Paco tiene ahora en su cuenta -50'
```


Observa que la cantidad es positiva, por lo que el programa realiza los cálculos. Este resultado no tiene sentido y es un indicador de que no hay una excepción para evitar este error. Estos son los detalles finos que se deben cuidar con el fin de que el programa funcione de manera correcta.

Para terminar la parte de la herencia, es posible verificar si las clases Gasto y Ahorro son clases hijo o subclases de Cuenta

```
print(issubclass(Ahorro,Cuenta))
True
print(issubclass(Gasto,Cuenta))
True
print(issubclass(Gasto,Ahorro))
False
```

En este último caso, naturalmente, Cuenta y Ahorro funcionan de manera independiente, aunque tengan una misma clase origen.

Hasta ahora, los métodos empleados utilizan la palabra reservada self, pero hay métodos que hacen uso de otras palabras reservadas. El uso de otras palabras reservadas implica que estás utilizando un método diferente al que ya haz utilizado. Python define tres tipos de métodos

- Métodos de clase
- Métodos estáticos
- Método

10.6 Métodos de clase

Al utilizar este método se antepone el método @classmethod. Una característica importante es que, al invocarlo, no es necesario crear el objeto. El siguiente ejemplo muestra los números telefónicos de emergencia de tres países además del tipo de emergencia que atienden.

```
class Emergencia:
    def __init__(self,tipo):
self.tipo=tipo def __repr__(self):
return f'Emergencia({self.tipo !r})'
    @classmethod
    def Colombia(cls):
        return cls(['Accidente','Policía','Marque 123'])
    @classmethod
    def Salvador(cls):
        return cls(['Accidente','Policía','Marque 911'])
    @classmethod
    def Paraguay(cls):
        return cls (['Bomberos', 'Policía', 'Marque 131'])
print(Emergencia.Paraguay())
Emergencia(['Bomberos', 'Policía', 'Marque 131'])
```

En este tipo de métodos utiliza la palabra reservada clc en lugar de self. Observa que no hay definido un objeto, por lo que el método trabaja con los valores definidos entre las distintas clases.

El siguiente script muestra la forma en que trabaja el método de instancia y el método estático.

```
class Emergencia:
    def __init__(self,emergencia,numero):
self.emergencia=emergencia self.numero=numero
def __repr__(self):
return(f'Emergencia({self.emergencia}, 'f'{self.numero})')
def pais(self):
return self.pais_emergencia(self.numero)
    @staticmethod
def paises(A):
return f'Costa Rica, Ecuador, El Salvador, E.U., México'
nueva_emergencia=Emergencia(['Policía','Ambulancia','Bomberos'],911)
```

```
print(nueva_emergencia.emergencia)
['Policía', 'Ambulancia', 'Bomberos'] print(nueva_emergencia.numero)
911
print(nueva_emergencia.paises(911))
Costa Rica, Ecuador, El Salvador, E.U., México
```

En las líneas ¿? El código es muy parecido al habitual que ya haz utilizado. En esta parte, el ejemplo muestra el tipo de emergencia que puede atenderse cuando se llama al número 911. En la segunda parte se utiliza un método estático que trabaja de manera independiente a la primera parte. Cuando se invoca, aparecen algunos países que emplean al número 911 como número de emergencia. Este número, aunque es el mismo que en el método de instancia, refieren a dos tareas diferentes. De aquí que se haya empleado el método estático.

10.7 Polimorfismo

El polimorfismo se define como la capacidad que tienen los objetos, en diferentes clases, para usar un comportamiento o un atributo con el mismo nombre, pero con diferente valor. Este es un ejemplo sencillo.

```
class Matematicas: num_creditos = 8 def creditos(self):
        print('Matemáticas tiene 8 créditos')
class Idioma: num_creditos = 5 def creditos(self):
        print('Idioma tiene 5 créditos')
#Hay un objeto con diferentes valores
#Hay dos clases con el mismo atributo
```

Para probar el código, se crea un objeto, en este caso asignatura

```
asignatura=Matematicas()
asignatura.creditos()
```


Observa que se están definidas dos clases, donde los métodos tienen el mismo nombre, también hay dos variables con el mismo nombre, pero con diferente valor. Esto es polimorfismo. A pesar de las similitudes, el objeto toma solo una opción de acuerdo a una clase específica.

En este caso donde hay dos métodos con el mismo nombre se le llama polimorfismo por método. También se puede dar por función o por herencia. A continuación, se muestra un ejemplo de polimorfismo por herencia.

```
class Pais:
    def moneda(self):
        print('Cada país tiene una moneda propia')
class honduras(Pais):
    def moneda(self):
        print('En Honduras, la divisa es la Lempira')
class nicaragua(Pais):
    def moneda(self):
        print('En Honduras, la divisa es el Córdova')
class paraguay(Pais):
    def moneda(self):
        print('En Paraguay, la divisa es el Guaraní')
```

Para mostrar los resultados

```
divisa=Pais()
divisa.moneda()
```

Cada país tiene una moneda propia

```
divisa_nicaragua=nicaragua()
divisa_nicaragua.moneda()
```

En Nicaragua, la divisa es el Córdova

Observa que hay tres subclases o clases hijo, y en cada una de ellas los métodos tienen el mismo nombre.

Dentro de las librerías definidas en Python se encuentra commissionemployee, esta librería resulta útil para administrar listas de empleados y resulta un ejemplo común, por sus características, en diversos libros y páginas web dedicadas a la programación en Python.

Enseguida se muestra un ejemplo sencillo de la forma en que se emplea. Las clases, variables y atributos están escritos en inglés, más aún, los comentarios están escritos en español. En este ejemplo se muestra el caso de dos tipos de empleados, uno que recibe un pago por comisión (commission_rate) en ventas (gross_sales) de y otro que también recibe la comisión, pero que también cuenta con un salario base (base_salary)

```
"""Pago a empleado por comisión."""
from commissionemployee import CommissionEmployee
from decimal import Decimal
```

En esta parte se definen las clases que se importan de las librerías commissionemployee y decimal respectivamente.

```
class SalariedCommissionEmployee(CommissionEmployee):
      def __init__(self, first_name, last_name, gross_sa-
      les, commission_rate, base_salary):
"""Atributos para un empleado que gana comisión."""
      super ().__init__ (first_name, last_name, gross_sa-
      les, commission_rate) self.base_salary= base_salary
      # validate via property
```

Aquí se define la clase SalariedCommissionEmployee y los atributos de los objetos. El método super(), se utiliza para llamar métodos definidos y en la herencia múltiple. Este método se utiliza solo en subclases y se caracteriza por brindar más atención a tales subclases.

```
@property
def base_salary(self): return self._base_salary
@base_salary.setter
def base_salary(self, salary):
"""Escribe el salario base o un ValueError si es inváli-
do.""" if salary < Decimal ('0.00'):
raise ValueError ('El salario base debe ser >= to 0')
self._base_salary = salary
```

En estas líneas puedes observar dos clases con el mismo nombre (polimorfismo). La primera está precedida de @property. Esto indica que la clase `base_salary` definida es de solo lectura cuando clase `base_salary` ya tiene asignado un valor. Si se va a ingresar un nuevo valor para clase `base_salary`, se debe utilizar la segunda clase. La segunda está precedida por @base_salary. setter. Python define dos tipos de métodos uno llamado de adquisición u obtención (getter) y otro llamado de colocación (setter). Esto significa que la clase `base_salary` debe proporcionar (colocar) un valor a la variable. Observa que este método valida el salario base, si no es válido muestra una excepción del tipo ValueError.

```
def earnings(self):
"""Calculate earnings."""
return super ().earnings() + self.base_salary
```

Aquí se definen la clase earnings, esta es la ganancia del empleado debida al tanto por ciento de la comisión. Este método regresa el salario más la comisión ganada.

```
def __repr__(self):
"""Muestra la cadena para repr()."""
return ('Salaried' + super ().__repr__ () +
f'\nsalario base : {self.base_salary:.2f}')
```

Esta clase utiliza el método __repr__. A diferencia de __str__, el método __repr__.es una representación formal de un string (cadena) para Python. __str__ está más orientado a ser entendido por los usuarios, mientras que __repr__ está más orientado a ser interpretado por Python. Esta última característica es útil a los programadores cuando requieren depurar sus programas. La tarea realizada en estas líneas, es la de calcular el salario final del empleado.

A continuación, se muestra un ejemplo de la manera en que funciona este programa

```
s = SalariedCommissionEmployee('Eva', 'García', Decimal
('5000.00'), Decimal ('0.1'), Decimal('3000.00'))
print(s.first_name,s.last_name,s.ssn,s.gross_sales,s.com-
mission_rate,s.base_salar y)
print(f'{s.earnings():0.2f}')
Eva García ABCD800808 5000.00 0.1 3000.00
3500.00
```

CAPÍTULO 11

Recursiones, iteraciones, búsqueda y ordenamiento

11.1 Introducción

En este capítulo se tratan tres temas clásicos de programación recursión-iteración, búsqueda y ordenamiento. A pesar del tiempo que ha pasado desde que se empezaron a utilizar las computadoras, estos temas siguen siendo actuales y permiten al programador en formación, conocer la forma en que se realizan estos procedimientos.

Así que... trae tu bata y tu cubrebocas. Python ya tienen preparado el microscopio. Ponte atento que iniciaremos un análisis minucioso de la forma en que trabajan algunos algoritmos.

Suerte en el laboratorio de la programación.

11.2 Algoritmos Recursivos e Iterativos

Una sucesión de Fibonacci funciona de la siguiente forma.

La sucesión inicia con los primeros dos números $k_1 = 0$ y $k_2 = 1$

El tercero es la suma del segundo y del primero,

$$k3 = k2 + k1 = 1 + 0 = 1.$$

El cuarto es la suma del tercero y el segundo

$$k_4 = k_3 + k_2 = 1+1=2.$$

El quinto, es la suma del cuarto y del tercero

$$k_5 = k_4 + k_3 = 2+1=2.$$

Si seguimos esta regla, entonces

$$k_{100} = k_{99} + k_{98}.$$

Sea cual sea el valor de k_{99} y k_{98}. Más aún, para cualquier número natural n que sea mayor a 2, se cumple

$$k_n = k_{n-1} + k_{n-2}.$$

Por ejemplo, si $k = 100$, entonces $k - 1 = 99$ y $k - 2 = 98$. Como estaba escrito antes. La fórmula de k_n, es una fórmula de recursión y a diferencia de las fórmulas que aprendemos en las matemáticas del nivel básico, como

$$A = \pi r^2, \text{ o } F=ma$$

Donde las cantidades representaban valores físicos o geométricos, en una fórmula recursiva aparecen elementos del mismo tipo, es decir, n_k, n_{k-1}, n_{k-2} son todos números y tienen una variable ordenada de números naturales, por ejemplo, los subíndices k, $k-1$, $k-2$, de este ejemplo.

Ejemplo 1

La sucesión de Fibonacci.

```
"""ejemplo1_C11.py"""
def Fibonacci(n):
    if n in (0,1):
       return n
    else:
       return Fibonacci(n-1)+Fibonacci(n-2)
for n in range (31):
    print(f'k({n})={Fibonacci(n)}')
k(0)=0
k(1)=1
k(2)=1
k(3)=2
k(4)=3
k(5)=5
k(6)=8
k(7)=13
k(8)=21
```



```
⋮
k(27)=196418
k(28)=317811
k(29)=514229
k(30)=832040
```

Ejemplo 2

Series geométricas.

Las series geométricas finitas, son una suma de potencias de un número r, que pueden estar multiplicadas por otro número a. Estas series (sumas) tienen la forma

$$a+ar+ar^2 +ar^3 +\ldots+ar^{n-1} = a(1+r+r^2+r^3+\ldots+r^{n-1})$$

Esta serie se puede ver de una forma recursiva. Utilizaremos la variable s en lugar de k (por aquello de las sumas). Los elementos de la sucesión se describen de la siguiente manera

$$s_1 = a$$
$$s_2 = a + ar = s_1 + ar$$
$$s_3 = a + ar + ar^2 = s2 + ar^2$$
$$s_4 = a+ar+ar^2 +ar^3 =s_3 +ar^3$$

Entonces, para cualquier número natural n,

$$s_n = s_{n-1} + ar^{n-1}$$

El siguiente ejemplo es la construcción de una serie geométrica con $a=3$, $r=1/2$ y $n=20$

```
"""ejemplo2_C11.py"""
a=3
r=1/2
def serie_geom(n):
if n == 0: return n
else:
return serie_geom (n - 1) + a * r ** (n - 1)
for n in range (1, 10):
print (f'k({n})={serie_geom (n)}')
```

Los resultados son

```
s(1)=3.0
s(2)=4.5
s(3)=5.25
```

```
s(4)=5.625
s(5)=5.8125 s(6)=5.90625
⋮
s(15)=5.99981689453125
s(16)=5.999908447265625
s(17)=5.9999542236328125
s(18)=5.999977111816406
s(19)=5.999988555908203
s(20)=5.999994277954102
```

Con los valores asignados, la suma es

$$3(1+½+(½)^2+(½)^3+\cdots+(½)^{19})=5.999994277954102$$

De acuerdo al comportamiento de esta sucesión de valores, mientras más aumentas el número de términos, la suma se acerca cada vez más al valor de 6.

Ejemplo 3

Número factorial.
Otro elemento que se puede expresar de forma recursiva es el factorial de un número entero **n**, este número se denota como *n!* y se define como

$$n! = n(n-1)(n-2)(n-3)\cdots(3)(2)(1)$$

Esto es

$$1! = 1$$
$$2! = (2)(1) = 2$$
$$3! = (3)(2)(1) = 6$$
$$4! = (4)(3)(2)(1) = 24 \ldots etc$$

De manera recursiva, se puede escribir como

$$1! = 1$$
$$2! = (2)1! = 2$$
$$3! = (3)2! = 6$$
$$4! = (4)3! = 24 \ldots etc$$

De manera que para cualquier número natural **n**, se puede escribir

$$n! = n(n-1)!$$

En Python

```
"""ejemplo3_C11.py"""
def factorial(n): if n<=1:
return 1
return n*factorial(n-1)
for i in range(1,21): print(f'{i}!={factorial(i)}')
```

Los resultados de los primeros 20 números naturales son

```
1!=1
2!=2
3!=6
4!=24
5!=120
6!=720
⋮
17!=355687428096000
18!=6402373705728000
19!=121645100408832000
20!=2432902008176640000
```

Los dos ejemplos anteriores se han escrito de forma recursiva. Pero también es posible hacer esto de manera iteriativa. Por ejemplo, la sucesión de Fibonacci

```
"""ejemplo4_C11.py"""
def Fibonacci_iterativo(n): suma=0
aux=1
for j in range(0,n):
aux, suma=suma, suma+aux return suma
print(Fibonacci_iterativo(7))
13
```

Observa la forma en que se han implementado los valores iniciales para la serie de Fibonacci. A uno se le ha llamado suma y al segundo se le llama aux. Dentro del ciclo para cada valor de j, los valores aux, suma y suma+aux van desplazándose a la izquierda.

Cuando j = 0, aux = 1, suma = 0, suma + aux = 1
Cuando j = 1, aux = 0, suma = 1, suma + aux = 1
Cuando j = 2, aux = 1, suma = 1, suma + aux = 2
Cuando j = 3, aux = 1, suma = 2, suma + aux = 3

Y para calcular los números factoriales

```
"""ejemplo5_C11.py"""
factorial=1
for n in range(20,1,-1):
factorial *= n
print(factorial)
2432902008176640000
```

Desde que estos algoritmos se han estudiado, prevalece el debate en la eficiencia y la adecuada estructura del algoritmo Al hacerlo de forma recursiva el código es más pequeño, más elegante y

la forma de escribir el código obedece casi de forma directa a las fórmulas de recurrencia, haciéndolo fácil de programar. Sin embargo, para efectos de mantener una programación más depurada y eficiente con el uso de memoria y almacenamiento de datos, los métodos iterativos suelen ser mejores.

Otro par de tópicos clásicos de programación son la búsqueda y el ordenamiento. A continuación, se trata cada uno de estos tópicos.

11.3 Búsqueda

En esta sección se presentan dos métodos de búsqueda. Uno que es muy sencillo de programar y otro que es mucho más rápido, pero también, más complejo de programar.

El primero de ellos es el algoritmo de búsqueda lineal. Este método busca un elemento dentro de un arreglo, por lo prueba con cada elemento del arreglo si existe alguna coincidencia. Si esto ocurre o no, informa al usuario el resultado de la búsqueda.

Para probar este método considera el siguiente conjunto de valores

```
1.70, 1.65, 1.58, 1.81, 1.84, 1.78, 1.68, 1.72
```

Este método es llamado de búsqueda lineal, y se presenta a continuación.

```
def busqueda_lineal(datos, dato_buscar):
    for indice, valor in enumerate (datos):
        if valor == dato_buscar: return indice
    return -1
datos = [1.70, 1.65, 1.58, 1.81, 1.84, 1.78, 1.68, 1.72]
print('El número que buscas tiene índice:',busqueda_li-
neal(datos, 1.81))
El número que buscas tiene índice: 3
```

Observa la rutina. Este programa revisa cada valor de la lista, si lo encuentra, regresa el índice de la lista donde se encuentra el valor. Si no lo encuentra regresa -1.

Cuando el valor de búsqueda se ubica en la posición *n* de la lista, se hacen *n* comparaciones. Pero si el valor no está, se realizan tantas comparaciones como elementos tenga la lista. Este último detalle es llamado el peor de los casos para determinar la eficiencia del algoritmo. El algoritmo de búsqueda lineal es poco eficiente, aunque fácil de programar. El algoritmo es rápido, cuando la lista no es muy grande, pero puede ser tardado cuando la cantidad de elementos es considerablemente grande.

El otro método es llamado búsqueda binaria. Aquí se muestra una descripción del método.

- Considera de manera inicial a toda la lista ordenada, en orden ascendente, por ejemplo.
- Revisa el valor que se encuentra en la mitad de la lista (el valor central), si es el valor que buscas,regresa el índice de este valor.

- Si no es el valor central y el valor buscado es menor, descarta el conjunto de valores que están a la derecha, incluyendo el punto medio.
- Si no es el valor central y el valor buscado es mayor, descarta el conjunto de valores que están a la izquierda, incluyendo el punto medio.
- Después de descartar los segmentos que no son de interés, revisa el segmento restante, con los mismos criterios.
- Si resulta que algún segmento ya no tiene elementos, significa que el valor que buscas no está presente en la lista.

A continuación, se muestra un código más elaborado que realiza la búsqueda binaria.

```
"""Implementación de la búsqueda binaria."""
import numpy as np
def busqueda_binaria(datos, dato_buscar): bajo = 0
alto = len (datos) - 1
medio = (bajo + alto + 1) // 2 # índice del elemento medio
localizacion = -1
```

En esta parte el programa divide al conjunto en dos segmentos, cuando ya no sea posible mostrará -1 para indicar que el elemento no está en la lista

```
while bajo <= alto and localizacion == -1:
    print (elementos_restantes (datos, bajo, alto))
    print (' ' * medio, end='')
    print (' * ')
```

En esta parte, el programa muestra los elementos que quedan y coloca un asterisco en la posición medio.

```
if dato_buscar == datos[medio]:
   localizacion = medio
elif dato_buscar < datos[medio]:
   alto = medio - 1
else:
   bajo = medio + 1
```

Cuando el elemento está en posición medio, asigna la localización al índice medio. Si no está y su valor es menor, elimina la mitad superior. Si no está, y su valor es mayor elimina la mitad inferior.

```
medio = (bajo + alto + 1) // 2
return localizacion
```

El programa recalcula la posición medio, si encontró al valor de búsqueda regresa la localización del dato.

```
def elementos_restantes(datos, bajo, alto):
return ' ' * bajo + ' '.join (str (s) for s in datos[ba-
jo:alto + 1])
```

En esta parte, muestra los elementos restantes para la búsqueda.

```
def main():
datos = np.random.randint (10, 91, 15) datos.sort ()
print (datos, '\n')
buscar_key = int (input ('Ingresa un valor entero (-1 para
salir): '))
```

El programa crea un arreglo de prueba, de 15 números aleatorios enteros, en el intervalo [10, 90] y solicita al usuario el número que requiere buscar.

```
while buscar_key != -1:
localizacion = busqueda_binaria (datos, buscar_key)
if localizacion == -1:
print (f'{buscar_key} no se encuentra\n')
        else:
            print (f'{buscar_key} está en la posición {lo-
calizacion}\n')
buscar_key = int (input ('Ingresa un valor entero (-1 para
salir): '))
```

Repite el ciclo de búsqueda hasta ingresar -1.

```
if __name__ == '__main__':
    main ()
```

Esta parte del programa hace que el programa se ejecute y muestre los resultados. Algunas pruebas son las siguientes.

```
[10 14 21 22 35 45 50 60 61 64 68 82 83 83 88]

Ingresa un valor entero (-1 para salir): 14
10 14 21 22 35 45 50 60 61 64 68 82 83 83 88
                      *
10 14 21 22 35 45 50
          *
10 14 21
    *
14 está en la posición 1

Ingresa un valor entero (-1 para salir):  90
10 14 21 22 35 45 50 60 61 64 68 82 83 83 88
                      *
                        61 64 68 82 83 83 88
                                 *
                                   83 83 88
                                       *
                                         88
                                          *
90 no se encuentra
Ingresa un valor entero (-1 para salir): -1
```


Este programa está lleno de detalles que permiten ver como trabaja el método, por lo que buena cantidad de código se utiliza para mostrar tales detalles en la rutina. Cada segmentación en dos partes del conjunto de datos equivale a una comparación. Por lo tanto, un arreglo con 1,048,575 elementos (2^{20} – 1), en el peor de los casos, requiere de un máximo de 20 comparaciones para encontrar el valor de búsqueda. Esta es la gran diferencia entre la eficiencia del algoritmo de búsqueda lineal y el de búsqueda binaria.

11.4 Ordenamiento

El ordenamiento de datos es una de las aplicaciones más importantes en la programación. En esta sección se discuten tres métodos, dos de ellos sencillos, pero poco eficientes para una gran cantidad de datos. El tercero es más eficiente pero también más complicado de programar.

Para iniciar, observa esta lista que se requiere ordenar en orden ascendente

```
15 13 9 10
```

Una posible forma de ordenarla, es buscar el elemento más pequeño, y ponerlo al inicio, pero intercambiando las posiciones entre el elemento que estaba en la primera posición y el elemento más pequeño. Entonces la lista toma la forma

```
9 13 15 10
```

Ahora se repite el procedimiento descartando la primera posición. Es decir, se intercambian las posiciones de 13 y 10, entonces el resultado es

```
9 10 15 13
```

Ahora se repite el procedimiento descartando las primeras dos posiciones. El orden final es

```
9 10 13 15
```

A este proceso se le conoce como `ordenamiento por selección`. Un programa que realiza esta tarea es el siguiente.

```
def ordenamiento_sel(datos):
    for indice1 in range(len(datos) - 1):
        peque = indice1
```

En este ciclo, se reduce la cantidad de elementos, una vez que se ha encontrado el elemento más pequeño y se ubica en la primera posición.

```
for indice2 in range(indice1 + 1, len(datos)):
    if datos[indice2] < datos[peque]:
        peque = indice2
datos[peque], datos[indice1] = datos[indice1], datos[peque]
```

En esta parte se identifica el elemento más pequeño, entre los elementos que quedan, y se intercambia de posición con el elemento que tiene el índice más bajo. Con la última línea se arman los segmentos de la nueva lista ordenada.

Aplicaremos este algoritmo con una lista que contiene la estatura de 9 personas.

```
1.85, 1.70, 1.58, 1.83, 1.80, 1.75, 1.70, 1.90, 1.65
```

```
datos = np.array ([1.85, 1.70, 1.58, 1.83, 1.80, 1.75, 1.70,
1.90, 1.65])
print (f'Arreglo sin orden: {datos}\n')
Arreglo sin orden: [1.85 1.7 1.58 1.83 1.8 1.75 1.7 1.9 1.65]
ordenamiento_sel (datos)
print (f'\nArreglo ordenado: {datos}\n')
Arreglo ordenado: [1.58 1.65 1.7 1.7 1.75 1.8 1.83 1.85 1.9 ]
```

Realizar esta tarea de forma manual, proporcionaría los siguientes resultados.

Arreglo sin orden:

```
[1.85 1.7 1.58 1.83 1.8 1.75 1.7 1.9 1.65]
paso 1: 1.58 1.7 1.85 1.83 1.8 1.75 1.7 1.9 1.65
paso 2: 1.58 1.65 1.85 1.83 1.8 1.75 1.7 1.9 1.7
paso 3: 1.58 1.65 1.7 1.83 1.8 1.75 1.85 1.9 1.7
paso 4: 1.58 1.65 1.7 1.7 1.8 1.75 1.85 1.9 1.83
paso 5: 1.58 1.65 1.7 1.7 1.75 1.8 1.85 1.9 1.83
paso 6: 1.58 1.65 1.7 1.7 1.75 1.8 1.85 1.9 1.83
paso 7: 1.58 1.65 1.7 1.7 1.75 1.8 1.83 1.9 1.85
paso 8: 1.58 1.65 1.7 1.7 1.75 1.8 1.83 1.85 1.9
Arreglo ordenado:
[1.58 1.65 1.7 1.7 1.75 1.8 1.83 1.85 1.9 ]
```

Supongamos nuevamente que se requiere ordenar en orden ascendente

```
15 13 9 10
```

Otra forma de ordenarla, es comparar los primeros dos elementos si el segundo es menor que el primero, se intercambian de lugar. Entonces la lista toma la forma

```
13 15 9 10
```

Ahora se compara el tercer elemento con los dos primeros. Pueden ocurrir tres casos diferentes:

- Que el tercer elemento se menor que los dos primeros, entonces el elemento se inserta en la primera posición (como en este ejemplo).

```
9 13 15 10
```


- Que el tercer elemento sea mayor que el primero y menor que el segundo, entonces el elemento se inserta en la segunda posición.
- Que el tercer elemento se mayor que los dos primeros, entonces el elemento se queda en la tercera posición, y se repite la rutina con el siguiente elemento.

Para finalizar, se compara el último elemento con los tres primeros, que ya están ordenados. El arreglo final es

```
9 10 13 15
```

A este proceso se le conoce como ordenamiento por inserción. Un programa que realiza esta tarea es el siguiente.

```
def ordenamiento_inser(datos):
    for n in range (1, len (datos)):
insertar = datos[n] mover_dato = n
```

En este ciclo, se reduce la cantidad de elementos, una vez que se han comparado elementos sucesivos, el elemento actual se inserta en la posición respectiva.

```
while mover_dato > 0 and datos[mover_dato - 1] > insertar:
datos[mover_dato] = datos[mover_dato - 1]
mover_dato -= 1
datos[mover_dato] = insertar
```

Este ciclo busca el lugar adecuado para colocar el elemento actual. Recuerda que en el ejemplo había tres posibilidades. Con la última línea, se arma el nuevo arreglo y se continua con el siguiente valor en la lista, cuando el ciclo for no ha terminado.

Aplicaremos este algoritmo a la lista de estaturas.

```
1.85, 1.70, 1.58, 1.83, 1.80, 1.75, 1.70, 1.90, 1.65
```

```
datos = np.array ([1.85, 1.70, 1.58, 1.83, 1.80, 1.75, 1.70,
1.90, 1.65]) print (f'Arreglo sin orden: {datos}\n')
Arreglo sin orden: [1.85 1.7 1.58 1.83 1.8 1.75 1.7 1.9 1.65]
ordenamiento_inser (datos)
print (f'\nArreglo ordenado: {datos}\n')
Arreglo ordenado: [1.58 1.65 1.7 1.7 1.75 1.8 1.83 1.85 1.9 ]
```

Las etapas de esta tarea realizada manualmente son:

Arreglo sin orden:

```
[1.85 1.7 1.58 1.83 1.8 1.75 1.7 1.9 1.65]
Paso 1: 1.7 1.85 1.58 1.83 1.8 1.75 1.7 1.9 1.65
Paso 2: 1.58 1.7 1.85 1.83 1.8 1.75 1.7 1.9 1.65
Paso 3: 1.58 1.7 1.83 1.85 1.8 1.75 1.7 1.9 1.65
```

```
Paso 4: 1.58 1.7 1.8 1.83 1.85 1.75 1.7 1.9 1.65
Paso 5: 1.58 1.7 1.75 1.8 1.83 1.85 1.7 1.9 1.65
Paso 6: 1.58 1.7 1.7 1.75 1.8 1.83 1.85 1.9 1.65
Paso 7: 1.58 1.7 1.7 1.75 1.8 1.83 1.85 1.9 1.65
Paso 8: 1.58 1.65 1.7 1.7 1.75 1.8 1.83 1.85 1.9
```

Arreglo ordenado:

```
[1.58 1.65 1.7 1.7 1.75 1.8 1.83 1.85 1.9 ]
```

Otra vez se requiere ordenar en orden ascendente

```
15 13 9 10
```

Una forma alternativa de ordenarla, es dividir la lista en dos

```
15 13      9 10
```

Para entonces ordenar cada una de manera individual

```
15 13      9 10
```

Y luego comparar ambas listas para hacer una lista única más larga.

En este paso se comparan los elementos más pequeños de cada lista, y se comienzan a mezclar las dos listas los primeros dos elementos ordenados son

```
9 13
```

Luego compara el segundo elemento de la primera lista (15), con los dos primeros y vuelve a mezclar los elementos

```
9 13 15
```

Y para finalizar compara el segundo elemento de la segunda lista, para obtener

```
9 10 13 15
```

A este proceso se le conoce como ordenamiento por mezcla. Este procedimiento es más complejo, ya que, si la lista tiene más elementos, cada subarreglo se dividiría otros arreglos más pequeños hasta llegar a subarreglos de tamaño 2, para luego comenzar a mezclarse con los otros subarreglos. Un programa que realiza esta tarea es el siguiente.

```
import numpy as np
def ordenamiento_mezcla(datos):
arreglo_ordenado (datos, 0, len (datos) - 1)
```

Aquí se define el método ordenamiento_mezcla para utilizar el algoritmo de ordenamiento por mezcla


```
def arreglo_ordenado(datos, bajo, alto):
"""Split datos, sort subarrays and mezcla them into sorted array."""
if (alto - bajo) >= 1:
medio1 = (bajo + alto) // 2 medio2 = medio1 + 1
```

En esta parte el algoritmo se calculan los elementos que tendrá cada lista

```
# output split step
print (f'split:
print (f'
print (f'
{subarreglo (datos, bajo, alto)}')
{subarreglo (datos, bajo, medio1)}')
{subarreglo (datos, medio2, alto)}\n')
        arreglo_ordenado (datos, bajo, medio1)
        arreglo_ordenado (datos, medio2, alto)
```

Ahora el algoritmo secciona a la lista en dos partes. Si la lista tiene una cantidad par de elementos, ambas listas tienen el mismo tamaño. Si tiene una cantidad impar, a la segunda lista se le asigna un elemento más.

```
mezcla (datos, bajo, medio1, medio2, alto)
```

Con esta parte, el algoritmo mezcla dos listas que ya están ordenadas

```
def mezcla(datos, izq, medio1, medio2, der): izq_indice = izq
der_indice = medio2
indice_combinado = izq
mezclado = [0] * len (datos)
```

La tarea del método mezcla, es ordenar dos subarreglos ordenados en uno solo también ordenado

```
    print (f'mezcla:  {subarreglo (datos, izq, medio1)}')
    print (f'         {subarreglo (datos, medio2, der)}')
```

Estas líneas muestran los subarreglos antes de ser mezclados

```
while izq_indice <= medio1 and der_indice <= der: if da-
tos[izq_indice] <= datos[der_indice]:
mezclado[indice_combinado] = datos[izq_indice] indice_com-
binado += 1
izq_indice += 1
else:
mezclado[indice_combinado] = datos[der_indice] indice_com-
binado += 1
der_indice += 1
```

En este ciclo, el programa compara a los elementos por parejas y ubica a cada elemento en el lugar adecuado

```
if izq_indice == medio2:
mezclado[indice_combinado:der + 1] = datos[der_indice:der + 1]
else:
mezclado[indice_combinado:der + 1] = datos[izq_indice:medio1 + 1]
```

La sentencia if verifica si aún quedan elementos en las listas por asignarse a algún subarreglo

```
datos[izq:der + 1] = mezclado[izq:der + 1]
print (f' {subarreglo (datos, izq, der)}\n')
```

Cuando no hay más subarreglos, se realizan copias que muestran en pantalla, la forma en que se ordena cada sub arreglo.

```
def subarreglo(datos, bajo, alto):
temporal = ' ' * bajo
temporal += ' '.join (str (item) for item in datos[bajo:alto
+ 1]) return temporal
```

Esta parte de código añade espacios para que el momento de mostrarlos se vean alineados.

Para aplicar esta rutina a la lista de estaturas.

```
1.85, 1.70, 1.58, 1.83, 1.80, 1.75, 1.70, 1.90, 1.65
```

```
datos = np.array ([1.85, 1.70, 1.58, 1.83, 1.80, 1.75,
1.70, 1.90, 1.65])
print (f' Arreglo sin orden: {datos}\n')
Arreglo sin orden: [1.85 1.7 1.58 1.83 1.8 1.75 1.7 1.9
1.65]

ordenamiento_mezcla (datos)
separa: 1.85 1.7 1.58 1.83 1.8 1.75 1.7 1.9 1.65
        1.85 1.7 1.58 1.83 1.8
                           1.75 1.7 1.9 1.65

separa: 1.85 1.7 1.58 1.83 1.8
        1.85 1.7
             1.58

separa: 1.85 1.7 1.58
        1.85 1.7
             1.58

mezcla: 1.85 1.7
```



```
                1.85
                1.7

       mezcla: 1.85
               1.7
               1.7 1.85

       separa: 1.7 1.85
               1.58
               1.58 1.7 1.85

       mezcla: 1.83 1.8
               1.83
               1.8

       mezcla: 1.83
               1.8
               1.8 1.83

       separa: 1.58 1.7 1.85
               1.8 1.83
               1.58 1.7 1.8 1.83 1.85

       separa: 1.75 1.7 1.9 1.65
               1.75 1.7
               1.9 1.65

       separa: 1.75 1.7
               1.75
               1.7

       mezcla: 1.75
               1.7
               1.7 1.75

       separa: 1.9 1.65
               1.9
               1.65

       mezcla: 1.9
               1.65
               1.65 1.9

       mezcla: 1.7 1.75
               1.65 1.9
               1.65 1.7 1.75 1.9
```

```
mezcla: 1.58 1.7 1.8 1.83 1.85
        1.65 1.7 1.75 1.9
        1.58 1.65 1.7 1.7 1.75 1.8 1.83 1.85 1.9

print (f'\nArreglo ordenado: {datos}\n')

Arreglo ordenado: [1.58 1.65 1.7 1.7 1.75 1.8 1.83 1.85 1.9 ]
```

Algunas consideraciones con respecto a los algoritmos de ordenamiento.

- Los métodos de ordenamiento por selección y por inserción realizan operaciones y tiempos similares, por lo que también su eficiencia es similar.
- Comparados con el método de ordenamiento por mezclas quedan en amplia desventaja, cuando la cantidad de elementos por ordenar es grande.

CAPÍTULO 12

Procesamiento de Lenguaje Natural

12.1 Introducción

Todas las formas de comunicación que conoces como texto, audio, video, señales, etc; así como el tipo de lenguaje que utilizas, español, inglés, chino entre muchos otros son objetos de estudio del Procesamiento de Lenguaje Natural (Natural Languaje Processing, NLP).

El procesamiento lenguaje de natural se realiza sobre colecciones de texto extraídos de tweets, Facebook, conversaciones, películas entre muchos otros contextos o aplicaciones en donde existe comunicación. el lenguaje natural carece de precisión matemática, ya que los diferentes puntos de vista de las personas dificultan la comprensión del lenguaje natural. Por ejemplo, el entorno social y la cultura, son ejemplos de factores que pueden afectar la interpretación de alguna noticia o comentario.

Hablemos de programación. ¿Listo?

Dialoguemos, esperando compartir los significados del procesamiento del lenguaje natural.

12.2 Instalación del módulo TextBlob

Para instalar TextBlob, abre el Promp de Anaconda. Si utilizas Windows es importante que abras el Promp como administrador. Para hacerlo selecciona Anaconda Powershell Promp en el menú de inicio, luego ejecuta el siguiente comando

```
conda install -c confa-forge textblob
```

Una vez que se complete la instalación, ejecuta el siguiente comando para descargar NLTK corpora utilizado por TextBlob:

```
ipython -m textblob.download_corpora
```

Esta librería contiene entre otras cosas: WordNet para definición de palabras, sinónimos y antónimos, Movie Reviews para análisis de emociones y the Brown Corpus para etiquetar partes de oraciones.

12.3 TextBlob

TextBlob es la clase fundamental para NLP con el módulo textblob. El siguiente código crea tu primer TextBlob.

```
from textblob import TextBlob
texto = 'Y cuando despertó. Todo a su alrededor era luminoso'
blob = TextBlob(texto)
blob
TextBlob("Y cuando despertó. Todo a su alrededor era luminoso")
```

Entre las características de TextBlob se encuentran los métodos Sentences y Words, los cuales pueden ser comparados con arreglos (strings) y proporcionan otros métodos para tareas de NLP.

Para obtener una lista de oraciones del texto puedes utilizar la propiedad sentence para obtener una lista de objetos Sentence.

```
blob.sentences
[Sentence("Y cuando despertó."), Sentence("Todo a su alre-
dedor era luminoso")]
```

O también puedes obtener una lista de palabras.

```
blob.words
WordList(['Y', 'cuando', 'despertó', 'Todo', 'a', 'su',
'alrededor', 'era', 'luminoso'])
```

Hasta aquí no hay problemas con el lenguaje, sin embargo, las librerías de Python para modelar y reconocer emociones están diseñadas para trabajar con el idioma inglés. Así que utilizaremos frases y texto principalmente en este idioma.

```
texto = 'I have good discipline to study. In the future I
will be a great engineer'
blob = TextBlob(texto)
blob
TextBlob("I have good discipline to study. In the future I
will be a great engine er")
blob.sentences
[Sentence("I have good discipline to study."),
```



```
Sentence("In the future I will be a great engineer")]
blob.words
WordList(['I', 'have', 'good', 'discipline', 'to', 'study',
'In', 'the', 'future', 'I', 'will', 'be', 'a', 'great',
'engineer'])
```

Las propiedades sentences y word separan oraciones completas y palabras respectivamente. A esta acción le llamaremos tokenizar texto.

El etiquetado de partes en una oración (Parts-of-speech) es la tarea de evaluar palabras, basados en un contexto con el fin de identificar cada palabra como parte de una oración. En inglés hay 8 partes principales en una oración:

- Sustantivos
- Pronombres
- Verbos
- Adjetivos
- Conjunciones
- Adverbios
- Interjecciones
- Preposiciones

La propiedad tag regresa una lista que contiene una palabra y una cadena que indica la parte a la que pertenece.

```
blob.tags
[('I', 'PRP'),
('have', 'VBP'),
('good', 'JJ'),
('discipline', 'NN'),
('to', 'TO'),
('study', 'VB'),
('In', 'IN'),
('the', 'DT'),
('future', 'NN'),
('I', 'PRP'),
('will', 'MD'),
('be', 'VB'),
('a', 'DT'),
('great', 'JJ'),
('engineer', 'NN')]
```

Fase	Tarea
I	PRP, Pronombre personal
have	VBP, Verbo, presente que no es de tercera persona del singular
good, great	JJ, Adjetivo
discipline, future, engineer	NN, Sustantivo singular
to	TO
study, be	VB, Verbo forma básica
in	IN, Preposición o conjunción
the, a	DT, Determinate de la clase de una referencia
will	MD, Modal

La propiedad noun_phrases de TextBlob regresa una lista de objetos que contiene una lista de objetos Word, cada uno con una frase del enunciado.

```
blob.noun_phrases
WordList(['good discipline', 'great engineer'])
```

Uno de los grandes aportes al análisis de lenguaje, es la interpretación que Python hace de las emociones a través de propiedad sentiment de un objeto Sentiment.

```
blob.sentiment
Sentiment(polarity=0.5, subjectivity=0.4916666666666667)
```

La polaridad indica el tipo de sentimiento, negativo si es cercano a -1.0, positivo si es cercano a 1.0 y neutro cuando es 0.0. La subjetividad es un valor de 0.0, cuando es objetivo, hasta 1.0 cuando se clasifica como totalmente subjetivo. Otro ejemplo

```
from textblob import TextBlob
texto2= 'I am a bad student. I will not be able to finish
my studies' blob2 = TextBlob(texto2)
blob2.sentiment
Sentiment(polarity=    -0.09999999999999992,
          subjectivity= 0.6458333333333333  )
```

Es posible determinar el valor de polarity y subjetivity de cada oración individual. La estrategia es obtener una lista de objetos Sentence y entonces mostrar la propiedad sentiment.

```
for sentence in blob2.sentences: print(sentence.sentiment)
Sentiment(    polarity= -0.6999999999999998,
          subjectivity=  0.6666666666666666)
Sentiment(    polarity=  0.5,
          subjectivity=  0.625)
```


Por default Python utiliza las técnicas de la librería Pattern a través de PatternAnalyzer y de sus propiedades TextBlob, Sentences y Words. Sin embargo, TextBlob tiene otra librería llamada NaiveBayesAnalyzer, la cual se entrena sobre una base de datos de reseñas de películas.

Naive Bayes es un algoritmo bastante popular dentro de los algoritmos de aprendizaje de machine learning. Observa los detalles que ofrece, al aplicar esta librería

```
from textblob.sentiments import NaiveBayesAnalyzer blob2=-
TextBlob(texto2, analyzer=NaiveBayesAnalyzer())
blob2
TextBlob("I am a bad student. I will not be able to finish
my studies")
blob2.sentiment
Sentiment(classification='pos',  p_pos=0.9111144935259595,
p_neg=0.08888550647403913)
```

La evaluación de Naive-Bayes indica que en total el sentimiento es clasificado como positivo (classification='pos'). El objeto p_pos de Sentiment indica que TextBlob es 91.1% positivo y el objeto p_neg es 8.8% negativo

Para revisar cada oración debes aplicar la propiedad sentiment. Esto lo realizaste anteriormente

```
for sentence in blob2.sentences:
print(sentence.sentiment)
Sentiment(classification='neg',p_pos=0.36600302868341356,
p_neg=0.6339969713165864)
Sentiment(classification='pos',  p_pos=0.9466836548265999,
p_neg=0.05331634517339978)
```

Observa como la evaluación de Naive-Bayes clasifica a la primera oración como negativa y la segunda como positiva.

12.4 Detección de lenguaje y traducción

Un reto enorme en programación ha sido la traducción entre diferentes idiomas, sin embargo, la solución se ha dado a través del procesamiento de lenguaje natural y de inteligencia artificial. Esto ha hecho posible los servicios de traducción de Google (más de 100 idiomas) y de Bing (más de 66 idiomas).

Si requieres determinar el lenguaje que tiene un segmento de texto puedes utilizar el método detect_language.

```
texto3='Bonjour'
blob3=TextBlob(texto3)
blob3.detect_language()
'fr'
```

Para los tipos de lenguajes soportados, puedes utilizar la siguiente sintaxis

```
saludo=blob3.translate(to='es')
saludo
TextBlob("Buenos dias")

[] blob
[] TextBlob("I have good discipline to study. In the future
I will be a great engineer")
[] mifrase = blob.traslate(to='es')
[] mifrase
[]  TextBlob("Tengo buena disciplina para estudiar. En el
futuro seré un gran ingeniero")
```

Las inflexiones en el idioma, se entienden como las diferentes formas que toman las palabras, tales como singular y plural, o los tiempos en los verbos, o el aplicar los verbos a diferentes personas. Cuando realizas el conteo de frecuencias de una palabra, es necesario tomar en cuenta este detalle, así que deberás convertir las inflexiones para que las palabras tengan una misma forma. Words y WordLists permiten convertir palabras en singular o plural. Observa este ejemplo.

```
[] from textblob import Word
[] sing_plur=Word('party')
[] sing_plur.pluralize() 'parties'
[] plur_sing=Word('lives')
[] plur_sing.singularize() 'life'
```

Realizar estas conversiones, resulta una tarea más complicada de poner o quitar la letra s.

```
[] from textblob import TextBlob
[] vegetales=TextBlob('potato tomato carrot').words
[] vegetales.pluralize()
WordList(['potatoes', 'tomatoes', 'carrots'])
```

Python también permite realizar una revisión ortográfica. El método spellcheck regresa una lista de posibles correcciones ortográficas y un valor de confianza que representa la posibilidad de cambio.

```
[] from textblob import Word
[] palabra=Word('whife')
[] %precision 2
'%.2f'
[] word.spellcheck()
[('while', 0.51),
 ('wife',  0.25),
 ('white', 0.24),
 ('whiff', 0.00),
 ('whine', 0.00)]
```

Observa que palabra con el más alto índice de ocurrencia no necesariamente es la palabra correcta para un contexto dado.

TextBlobs, Sentences y Words tienen el método correct, que puedes llamar para realizar las correcciones ortográficas. Este método regresa la palabra con el mayor valor de confianza.

```
[] palabra.correct()
'while'
```

Al aplicar correct a una oración, las funciones revisan la ortografía de cada palabra. Si existen errores, el método correct realiza los reemplazos necesarios.

```
[] from textblob import Word
[] enunciado=TextBlob('Yestarday was a bab dai')
[] enunciado.correct()
TextBlob("Yesterday was a bad day")
```

En Python, Stemming elimina un prefijo o sufijo de una palabra dejando solo una raíz, la cual puede o no, ser una palabra real. El método Lemmatization es similar, pero revisa los factores en la parte de la palabra y el significado, dando como resultado una palabra real.

Estas operaciones (Steamming y Lemmatization) son llamadas de normalización, y es una forma de preparar los datos para su posterior análisis. Otro complemento puede ser utilizar la raíz de la palabra y tratarlas todas con minúsculas.

Words y WordList permiten utilizar los métodos stem y lemmatize.

```
[] from textblob import Word
[] palabra2=Word('dormitories')
[] palabra2.stem()
'dormitori'
[] palabra2.lemmatize()
'dormitory'
```

Una de las estrategias para verificar la similaridad entre documentos, es comparar la frecuencia de las palabras. Para hacer esto, es posible utilizar el módulo pathlib de Path class, de la librería estándar de Python.

```
from pathlib import Path
from textblob import TextBlob
from textblob import Word
texto=open('dracula.txt',encoding='utf-8')
libro_drac=TextBlob(texto.read())
libro_drac.words.count('crucifix')
19
[]libro_drac.words.count('Dracula')
41
[] libro_drac.words.count('blood')
108
```

Si el archivo de texto ya está tokenizado, puedes contar frases específicas en la lista a través del método count.

```
[] blob.noun_phrases.count('lady capulet')
[]
```

Además de permitir contar frases o palabras, Python utiliza la base de datos de WordNet a través de TextBlob, la librería NLTK y la interfaz de WordNet. Juntos permiten a los usuarios buscar definiciones y obtener sinónimos y antónimos.

Por ejemplo, si requieres buscar alguna definición:

```
[] from textblob import Word
word_test=Word('worker')
```

La propiedad definitions de la clase Word regresa una lista de todas las definiciones de palabra en la base de datos de WordNet.

```
word_test.definitions
['a person who works at a specific occupation',
'a member of the working class (not necessarily employed)',
'sterile member of a colony of social insects that forages
for food and cares for the larvae',
'a person who acts and gets things done']
```

Para encontrar un sinónimo se utiliza la propiedad synsets de Words. El resultado es una lista de objetos de Synset.

```
word_test.synsets
[Synset('worker.n.01'),
Synset('proletarian.n.01'),
Synset('worker.n.03'),
Synset('actor.n.02')]
```

Cada Synset representa un grupo de sinónimos. En la notación `worker.n.01`:

- `worker` es la forma lematizada de worker (en este caso la misma).
- `n` es la parte de la oración, en este caso a es de adjetivo, n por sustantivo, v de verbo, r de adverbio o s para adjetivo satélite.
- `01` es un índice numérico. Muchas palabras tienen múltiples significados. Éste el índice del correspondiente significado de Word Net.

Las palabras de paro o stop words son palabras que aparecen comúnmente en textos, sin embargo, estas palabras no aportan información útil al análisis de texto. Por lo que comúnmente se eliminan del análisis.

La librería NLTK tiene una lista de palabras Stop Words para diversos idiomas. Para poder visualizar esta lista, primero debes descargarla. Esto es posible con la función download del módulo nltk.


```
import nltk.
nltk.download('stopwords')
[] C:\users\equipment
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\marti\AppData\Roaming\nltk_data...
[nltk_data] Unzipping corpora\stopwords.zip.
True
```

Por ejemplo, para cargar la lista de palabras stop words del idioma inglés

```
from nltk.corpus import stopwords
stops=stopwords.words('english')
print(stops)
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'our-
selves', 'you', "you're", "you've", "you'll", "you'd",
'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', 'she', "she's", 'her', 'hers', 'herself',
'it', "it's", 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves', 'what', 'which', 'who', 'whom',
'this', 'that', "that'll", 'these', 'those', 'am', 'is',
'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an',
'the', 'and', 'but', 'if', 'or', 'because', 'as', 'un-
til', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'befo-
re', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
'in', 'out', 'on', 'off', 'over', 'under', 'again', 'fur-
ther', 'then', 'once', 'here', 'there', 'when', 'where',
'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only',
'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',
'will', 'just', 'don', "don't", 'should', "should've",
'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
"aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
"doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven',
"haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't",
'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "we-
ren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Ahora, debes crear un TextBlob del cual se remueven las palabras stop words

```
[] from textblob import TextBlob
[] blob=TextBlob('I have a beautiful day') Blob
TextBlob("I have a beautiful day")
```

Para finalizar, puedes borrar las palabras de paro, mediante una compresión de lista y añadir cada palabra a una lista nueva que no contenga palabras stop word.

```
[] [word for word in blob.words if word not in stops]
['I', 'beautiful', 'day']
```

12.5 Visualizando frecuencias de palabras

Con el libro de Drácula que ya haz utilizado, puedes descargar una lista de palabras frecuentes en el texto y visualizarlas por medio de una gráfica de Pandas. Aquí está el procedimiento.

```
from pathlib import Path
from textblob import TextBlob
texto=open('dracula.txt', encoding='utf-8')
libro_drac=TextBlob(texto.read())
```

Primero debes cargar las librerías necesarias y el libro que nos ha permitido realizar las tareas de análisis de texto.

```
from nltk.corpus import stopwords
stops=stopwords.words('english')
```

Luego cargar las palabras de paro de NLKT. Para obtener las primeras 15 palabras más repetidas.

```
items=libro_drac.word_counts.items()
```

El método ítems permite generar una tupla con una lista de palabras y las veces que se repite.

```
items=[item for item in items if item[0] not in stops]
```

Mediante una compresión de lista, se eliminan las tuplas que contienen palabras de paro.

```
from operator import itemgetter
items_ordenados=sorted(items,key=itemgetter(1),reverse=True)
```

Para determinar las primeras 15 palabras más frecuentes, se ordenan las tuplas en orden descendente por frecuencias. Aquí se emplea la función constructora sorted. Para especificar el elemento que ordena la tupla se emplea la función itemgetter del módulo operator.

Una vez que se han obtenido las palabras más repetidas, es posible seleccionar únicamente las primeras 15 de interés

```
primeros15=items_ordenados[1:16]
```

Para visualizar la lista de los primeros 15 utiliza un DataFrame de pandas

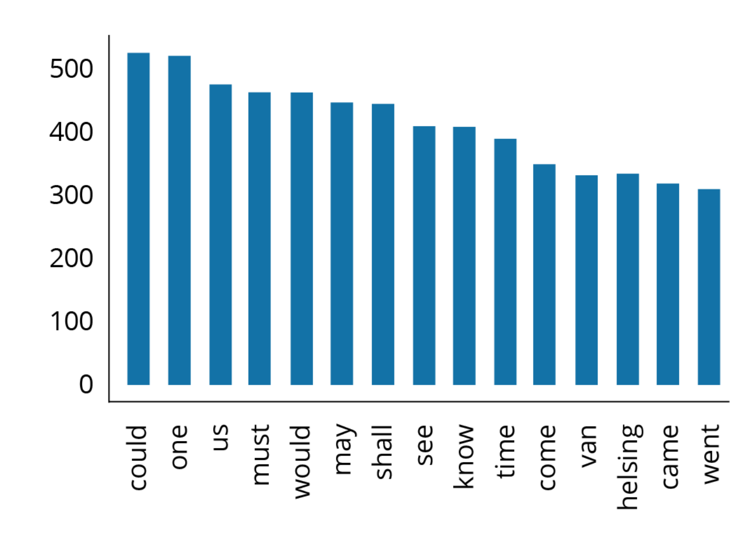
```
import pandas as pd datos=pd.DataFrame(primeros15,colum-
ns=['palabra','frecuencia'])
```


	Palabra	Frecuencia
0	could	509
1	one	507
2	us	463
3	must	451
4	would	447
5	may	433
6	shall	429
7	see	398
8	know	397
9	time	377
10	come	340
11	van	323
12	helsing	320
13	came	309
14	went	299

Para visualizar el comportamiento de los datos, utiliza un gráfico de barras

```
ejes=datos.plot.bar(x='palabra',y='frecuencia',legend=-
False)
```

Para obtener



Una de las bondades de Python es la facilidad de elaborar gráficos que pueden ser muy complicados. El siguiente procedimiento muestra la forma de generar una nube de palabras con el texto del libro de Drácula.

Inicialmente debes instalar desde el prompt de Anaconda la librería wordcloud

```
Conda install -c conda-forge wordcloud
```

En algunos casos será necesario instalarlo desde Anaconda Powershell Prompt, para utilizar los privilegios de administrador. Esta opción es necesaria, más aún si tu equipo de cómputo tiene varios usuarios.

Las librerías necesarias son

```
from pathlib import Path
from textblob import TextBlob
from wordcloud import WordCloud, STOPWORDS
```

Carga nuevamente el libro de Drácula y las palabras de paro.

```
texto=open('dracula.txt',encoding='utf-8').read()
stopwords=set(STOPWORDS)
```

Carga una imagen de máscara para la nube mediante la función imread. En este caso se ha seleccionado un óvalo.

```
import imageio
mascara=imageio.imread('mask_oval.png')
```

A continuación, se muestran algunas de las características específicas para la nube a crear

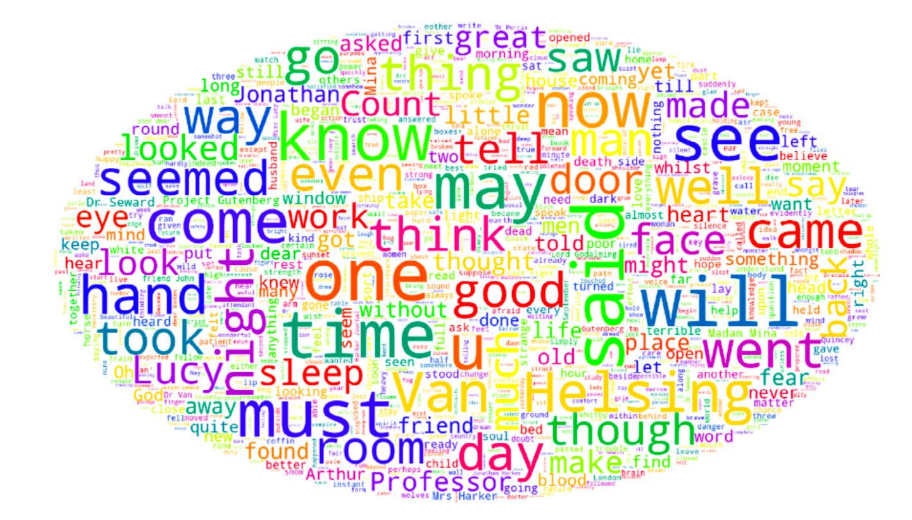
```
drac_nubpal=WordCloud(colormap='prism', background_color='whi-
te', max_words=1000, stopwords=stopwords, mask=mascara)
```

Luego se aplica el método generate de WordCloud. Este método tiene como argumento una cadena.

```
drac_nubpal.generate(texto)   <wordcloud.wordcloud.WordCloud   at
0x18291c7cd90>
Ya puedes obtener la nube
plt.imshow(drac_nubpal,interpolation='bilinear') plt.axis('off')
plt.show
```

Para guardar la imagen en tu equipo

```
drac_nubpal=drac_nubpal.to_file('dracula.png')
```

CAPÍTULO 13

Minería de datos

13.1 Introducción

Pío Baroja (1872-1956)

Fue un escritor español, que, aunque estudio medicina terminó por dedicarse a la literatura. Escribió novelas, cuentos cortos, poemas y obras teatrales. Su forma de escribir lo ubica en el terreno de la narrativa. Su estilo de vida apartado, abonó para clasificarlo como una persona solitaria y pesimista.

Algunas de sus obras más sobresalientes son:

- El árbol de la ciencia
- Zalacaín el aventurero
- Aventuras, inventos y mixtificaciones de Silvestre Paradox
- Camino de perfección
- Las inquietudes de Shanti Andía
- César o nada
- El aprendiz de conspirador
- Con la pluma y con el sable

Esta breve introducción a la vida y obra de Pío Baroja es una especie de carta de presentación, es un vistazo fugas del escenario que ya se vislumbra y que se concreta, en el análisis de algunas de sus obras.

Toma tu taza de café, algunas hojas y un buen bolígrafo. Escribamos en el idioma de Python

13.2 Minería de Datos

Wikipedia define minería de datos como:

"… un campo de la estadística y las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos".

Además, "supone aspectos de gestión de datos y de bases de datos, de procesamiento de datos, del modelo y de las consideraciones de inferencia, de métricas de intereses, de consideraciones de la teoría de la complejidad computacional, de post-procesamiento de las estructuras descubiertas, de la visualización y de la actualización en línea."

Esta última parte se puede resumir en tres partes principales:

Extracción y limpieza de la información *Exploración de los datos* *Modelado y visualización*

Para recorrer estas 3 etapas, utilizaremos 9 de las novelas de Pío Baroja. Las novelas fueron extraídas de la biblioteca del proyecto Gutenberg. Si requieres consultar la librería, puedes hacerlo en:

https://www.gutenberg.org/

Esta biblioteca virtual cuenta con más de 60,000 ebooks disponibles en forma gratuita y en diferentes formatos. La colección de libros fue descargada en formato de texto plano (txt) del enlace:

https://www.gutenberg.org/browse/authors/b#a2669

Las obras seleccionadas fueron:

Archivo	Nombre de la obra
pg61189	El amor, el dandysmo y la intriga
pg47103	El aprendiz de conspirador
pg60464	El árbol de la ciencia
49280-0	Los caminos del mundo
pg53517	Los caudillos de 1830
pg49470	Con la pluma y con el sable: Crónica de 1820 a 1823
pg51858	Los contrastes de la vida
pg48783	El escuadrón del brigante
pg50726	La ruta del aventurero

13.3 Etapa 1. Extracción y limpieza de la información

A continuación, se muestran las librerías necesarias para realizar el análisis exploratorio de las obras de Pío Baroja.

```
# Minería de datos en el análisis de obras literarias 2
#Librerías a utilizar
import matplotlib.pyplot as plt
import numpy as np
import os
import scipy
import math
from scipy.cluster.hierarchy import ward, dendrogram
from nltk.corpus import stopwords
from mpl_toolkits.mplot3d import Axes3D
from sklearn import manifold
from sklearn import metrics
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.manifold import MDS

```

- `scipy`, es una paquetería con varios toolbox orientados a atender problemas en cómputo científico. Los submódulos contienen aplicaciones como interpolación, integración, optimización, procesamiento de imágenes, estadística, funciones especiales, etc.
- `nltk`, es un conjunto de herramientas para el procesamiento de lenguaje natural.
- `mpl`, es un alias de la librería `pyplot` de matplotlib.
- `sklearn`, es una librería empleada para realizar análisis y minería de datos. Sus módulos atienden problemas de clasificación, regresión, cluster, reducción de dimensiones, selección de modelos y preprocesamiento de datos.
- `matplotlib`, es una biblioteca auxiliar en la generación de gráficos de datos contenidos en listas y arreglos.
- `numpy`, es una librería de Python especializada en el cálculo numérico y análisis de grandes bases de datos.
- `os`, este módulo permite realizar operaciones dependientes del sistema operativo como crear carpetas, listar contenidos de una carpeta, etc.
- `math`, es un módulo que permite utilizar funciones matemáticas.

Ahora se extraen los archivos de texto. Python brinda la posibilidad de descargar bases de datos de manera automática. En este caso, la información ya está descargada y tiene una ruta en el sistema operativo

`/Users/marti/textos`

Donde textos, es una carpeta que contiene los libros de Pío, en formato .txt, con decodificación utf-8.

```
20  #Parte 1. Obtención de información
21  path='/Users/marti/textos'
22  documents = []
23  titles = []
24  dirs = os.listdir(path)
25  for doc in dirs:
26  if doc.endswith('.txt'):
27  titles.append(doc)
28  f=open(os.path.join(path,doc),'r',encoding='utf-8')
29  words = f.read()
30  documents.append(words)
31  f.close()
32
```

En las líneas 21-24, Python ubica la ruta en donde se encuentran los libros y hace una lista de ellos. En el ciclo for en la línea 25, Python abre y lee cada libro (28-29) , para hacer un arreglo con todos los libros juntos (línea 30).

13.4 Parte 2. Preprocesamiento de la información

Como en el capítulo anterior, debes descargar las palabras de paro (stopwords) para quitarlas del análisis. El código permite a Python construir vectores y matrices con el esquema de frecuencia inversa. Esto significa que a las palabras con mayor frecuencia les asigna un valor menor, ya que estás tienen mayor importancia en el análisis. En la línea 38 se ajustan las dimensiones de las matrices, el método fit_transform realiza esta tarea.

```
33 #Parte 2. Preprocesar la información
34 sw=stopwords.words('spanish')
35
36 #Crea vectores sin stopwords y genera matriz tf-idf
37 tfidf_vectorizer = TfidfVectorizer(sw)
38 tfidf_matrix = tfidf_vectorizer.fit_transform(documents)
39 # tf idfd -->iniciales de term frecuency inverse document
   frecuency
40
41 #Se crea diccionario
42 diccionario = tfidf_vectorizer.get_feature_names()
43

C:\Users\marti\anaconda3\lib\site-packages\sklearn\utils\
validation.py:67: FutureW arning: Pass input=['de', 'la',
'que', 'el', 'en', 'y', 'a', 'los', 'del', 'se', 'las',
'por', 'un', 'para', 'con', 'no', 'una', 'su', 'al', 'lo',
'como', 'más', 'pero', 'sus', 'le', 'ya', 'o', 'este',
```



```
'sí', 'porque', 'esta', 'entre', 'cuando', 'muy', 'sin',
'sobre', 'también', 'me', 'hasta', 'hay', 'donde', 'quien',
'desde', 'todo', 'nos', 'durante', 'todos', 'uno', 'les',
'ni', 'contra', 'otros', 'ese', 'eso', 'ante', 'ellos',
'e', 'esto', 'mí', 'antes', 'algunos', 'qué', 'unos',
'yo', 'otro', 'otras', 'otra', 'él', 'tanto', 'esa', 'es-
tos', 'mucho', 'quienes', 'nada', 'muchos', 'cual', 'poco',
'ella', 'estar', 'estas', 'algunas', 'algo', 'nosotros',
'mi', 'mis', 'tú', 'te', 'ti', 'tu', 'tus', 'ellas', 'noso-
tras', 'vosotros', 'vosotras', 'os', 'mío', 'mía', 'míos',
'mías', 'tuyo', 'tuya', 'tuyos', 'tuyas', 'suyo', 'suya',
'suyos', 'suyas', 'nuestro', 'nuestra', 'nuestros', 'nues-
tras', 'vuestro', 'vuestra', 'vuestros', 'vuestras', 'esos',
'esas', 'estoy', 'estás', 'está' , 'estamos', 'estáis', 'es-
tán', 'esté', 'estés', 'estemos', 'estéis', 'estén', 'es-
taré', 'estarás', 'estará', 'estaremos', 'estaréis', 'es-
tarán', 'estaría', 'estarías', 'estaríamos', 'estaríais',
'estarían', 'estaba', 'estabas', 'estábamos', 'est abais',
'estaban', 'estuve', 'estuviste', 'estuvo', 'estuvimos',
'estuvisteis', 'estuvieron', 'estuviera', 'estuvieras',
'estuviéramos', 'estuvierais', 'estuvieran' , 'estuviese',
'estuvieses', 'estuviésemos', 'estuvieseis', 'estuviesen',
'estando', 'estado', 'estada', 'estados', 'estadas', 'es-
tad', 'he', 'has', 'ha', 'hemos', 'habéis', 'han', 'haya',
'hayas', 'hayamos', 'hayáis', 'hayan', 'habré', 'habrás',
'habrá', 'habremos', 'habréis', 'habrán', 'habría', 'ha-
brías', 'habríamos', 'habríais', 'habrían', 'había', 'ha-
bías', 'habíamos', 'habíais', 'habían', 'hube', 'hubiste',
'hubo', 'hubimos', 'hubisteis', 'hubieron', 'hubiera', 'hu-
bieras', 'hubiéramos', 'hubierais', 'hubieran', 'hubiese',
'hubieses', 'hubiésemos', 'hubieseis', 'hubiesen', 'ha-
biendo', 'habido', 'habida', 'habidos', 'habidas', 'soy',
'eres', 'es', 'somos', 'sois', 'son', 'sea', 'seas', 'sea-
mos', 'seáis', 'sean', 'seré', 'serás', 'será', 'seremos',
'seréis', 'serán', 'sería', 'serías', 'seríamos', 'seríais',
'serían', 'era', 'eras', 'éramos', 'erais', 'eran', 'fui',
'fuiste', 'fue', 'fuimos', 'fuisteis', 'fueron', 'fuera',
'fueras', 'fuéramos', 'fuerais', 'fueran', 'fuese', 'fue-
ses', 'fuésemos', 'fueseis', 'fuesen', 'sintiendo', 'senti-
do', 'sentida ', 'sentidos', 'sentidas', 'siente', 'sentid',
'tengo', 'tienes', 'tiene', 'tenemos', 'tenéis', 'tienen',
'tenga', 'tengas', 'tengamos', 'tengáis', 'tengan', 'ten-
dré', 'tendrás', 'tendrá', 'tendremos', 'tendréis', 'ten-
drán', 'tendría', 'tendrías' , 'tendríamos', 'tendríais',
'tendrían', 'tenía', 'tenías', 'teníamos', 'teníais', 'te-
nían', 'tuve', 'tuviste', 'tuvo', 'tuvimos', 'tuvisteis',
'tuvieron', 'tuviera' , 'tuvieras', 'tuviéramos', 'tuvie-
```

```
rais', 'tuvieran', 'tuviese', 'tuvieses', 'tuvié semos',
'tuvieseis', 'tuviesen', 'teniendo', 'tenido', 'tenida',
'tenidos', 'tenidas', 'tened'] as keyword args.
```

Esta son todas las palabras de paro. Aquí termina la parte de preprocesamiento de la información.

13.5 Parte 3. Análisis y visualización de los datos

En esta parte se verifican las dimensiones de las matrices

```
#Parte 3. Análisis y visualización de los datos
print('Corroborar tamaño de la matriz Documentos vs Términos')
print(tfidfmatrix.shape)

Corroborar tamaño de la matriz Documentos vs Términos
(9, 34320)
```

El 9, es la cantidad de libros y 34320 fue el ajuste que hizo Python para que todas las obras tengan las mismas dimensiones.

En esta parte se calcula el coeficiente se similitud coseno. Python tiene varios coeficientes, sin embargo, cosine_similarity es el que cuenta con más popularidad.

```
print('Obteniendo similitud de coseno entre 2 documentos
(si son iguales el valor es 1)')
cosine = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[2:3])
print(cosine)
print('Cálculo de distancia')
dist = 1 - cosine
print(dist)


Obteniendo similitud de coseno entre 2 documentos (si son
iguales el valor es 1) [[0.98095469]]
Cálculo de distancia
[[0.01904531]]
```

El coeficiente de similitud es cercano a 1, lo cual implica que las obras con índices 0 y 2 son muy parecidas.

La siguiente parte del código calcula el ángulo de separación entre los documentos. Para que se muestre mejor la relación entre los documentos, se realiza una conversión en grados.

```
print('Ángulo de separación de los documentos (grados)')
angle_in_radians = math.acos(cosine)
print(math.degrees(angle_in_radians))

print('Área de gráficos')
dist = 1 - cosine_similarity(tfidf_matrix)
```



```
print('Impresión de similitud de documentos por método de coseno')
np.round(dist, 2)

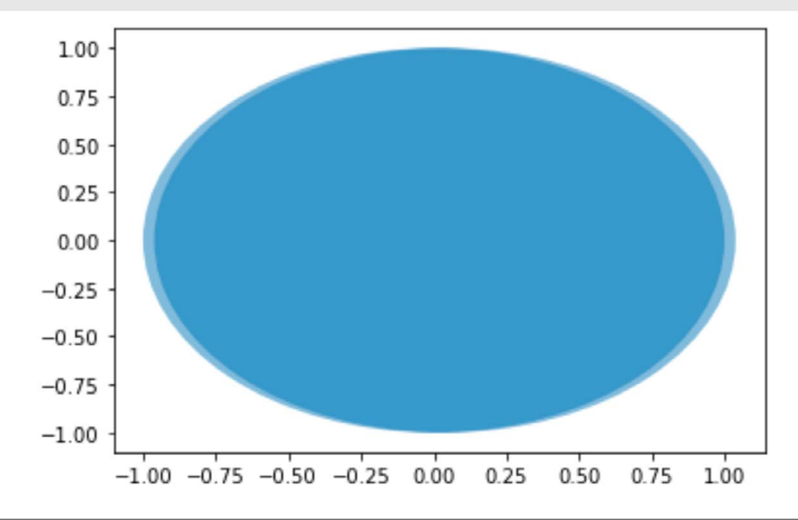

Ángulo de separación de los documentos (grados)
11.200136945764994
Impresión de similitud de documentos por método de coseno
array([[-0. , 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.04, 0.02],
       [ 0.02, 0. , 0.03, 0.02, 0.03, 0.03, 0.03, 0.04, 0.03],
       [ 0.02, 0.03, -0. , 0.02, 0.02, 0.03, 0.03, 0.05, 0.03],
       [ 0.02, 0.02, 0.02, 0. , 0.03, 0.04, 0.03, 0.04, 0.04],
       [ 0.02, 0.03, 0.02, 0.03, -0. , 0.02, 0.03, 0.04, 0.02],
       [ 0.02, 0.03, 0.03, 0.04, 0.02, -0. , 0.03, 0.05, 0.02],
       [ 0.02, 0.03, 0.03, 0.03, 0.03, 0.03, 0. , 0.04, 0.03],
       [ 0.04, 0.04, 0.05, 0.04, 0.04, 0.05, 0.04, -0. , 0.04],
       [ 0.02, 0.03, 0.03, 0.04, 0.02, 0.02, 0.03, 0.04, -0. ]])
```

El array resultante es una matriz de similaridades entre los diferentes documentos. La matriz que observas es una matriz cuadrada de dimensiones 9x9.

Aquí inicia la parte de visualización de los resultados. A continuación, se presenta un método visual para identificar la similaridad de dos obras de Pío.

```
r = 1
d = 2 * r * (1 - cosine)
circle1 = plt.Circle((0, 0), r, alpha=.5)
circle2 = plt.Circle((d, 0), r, alpha=.5)
## set axis limits
plt.ylim([-1.1, 1.1])
plt.xlim([-1.1, 1.1 + d])
fig = plt.gcf()
fig.gca().add_artist(circle1)
fig.gca().add_artist(circle2)

<matplotlib.patches.Circle at 0x225aa142850>
```



En este caso las dos circunferencias están prácticamente una sobre la otra, esto es un indicador de que ambas obras son muy similares.

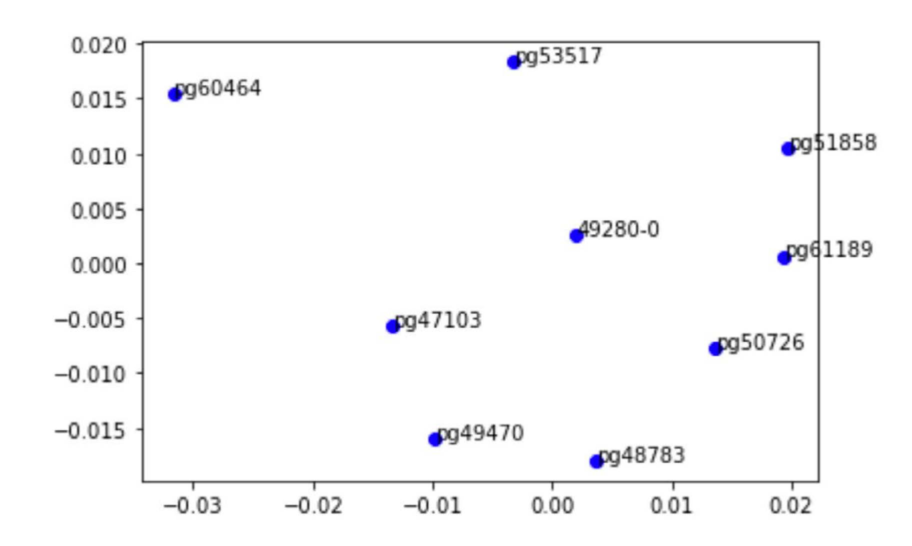
El análisis de cluster es una técnica del análisis multivariado que permite identificar grupos de datos, basados en la distancia que hay entre puntos de dimensión n. Visualmente puede generarse una gráfica que muestra la forma en que agrupan los individuos, o un dendograma que muestra gráficamente como es la distancia y el orden en que se agrupan los individuos.

La siguiente rutina muestra la dispersión de los individuos (obras de Pío Baroja)

```
print('Cluster de distancia entre documentos')
mds = MDS(n_components=2, dissimilarity="precomputed",
random_state=1)
pos = mds.fit_transform(dist)
xs, ys = pos[:, 0], pos[:, 1]
names = [os.path.basename(fn).replace('.txt', '') for fn
in titles]

for x, y, name in zip(xs, ys, names):
color = 'orange' if "d1" in name else 'blue'
plt.scatter(x, y, c=color)
plt.text(x, y, name)
plt.show()


Cluster de distancia entre documentos
```

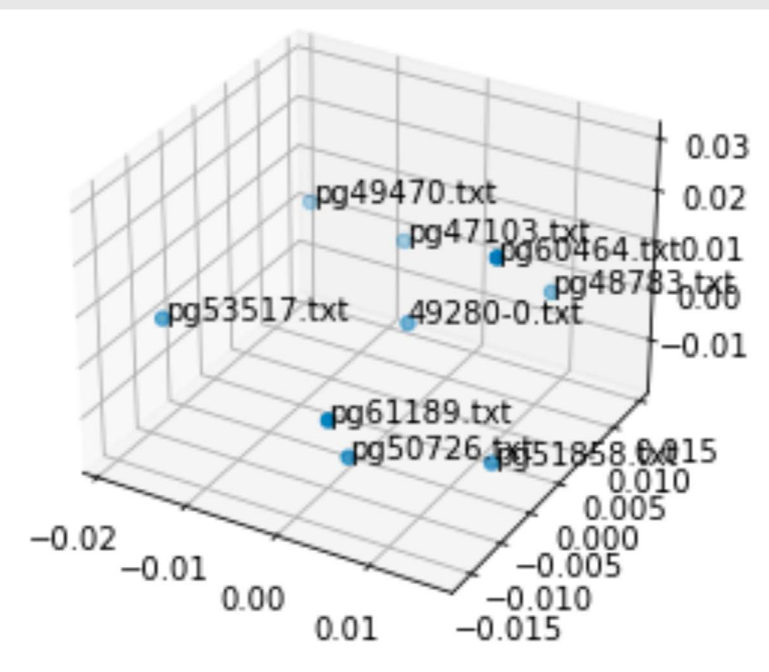


Observa que la obra pg60464 es la más alejada de las demás obras y que además, pg49470 y pg48783 son las más cercanas (más parecidas).

En esta parte se construye exactamente una gráfica como la anterior, pero en 3D


```
print('Cluster de documentos en 3D')
mds = MDS(n_components=3, dissimilarity="precomputed",
random_state=1)
pos = mds.fit_transform(dist)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(pos[:, 0], pos[:, 1], pos[:, 2])
for x, y, z, s in zip(pos[:, 0], pos[:, 1], pos[:, 2],
titles): ax.text(x, y, z, s)
plt.show()

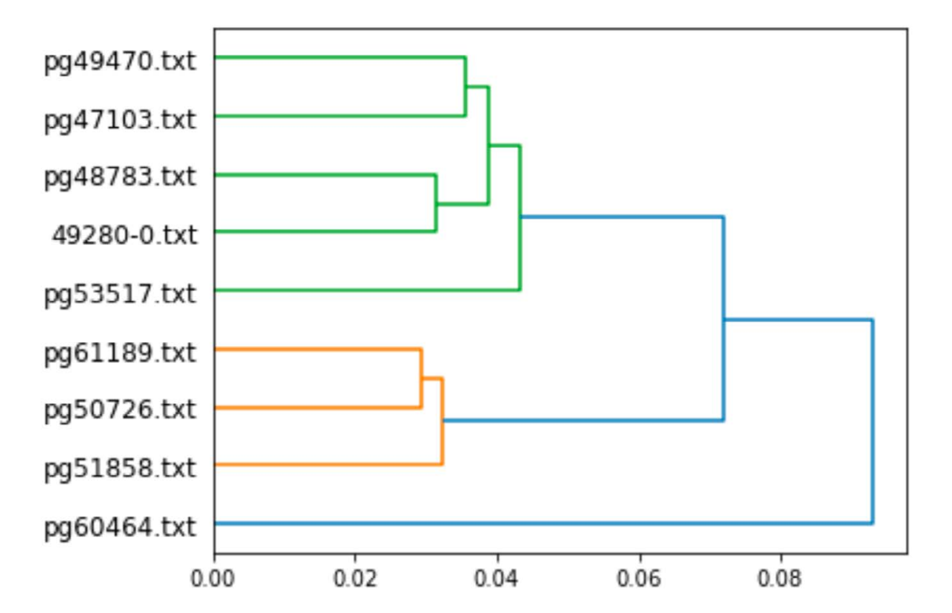

Clustering de documentos en 3D
```



La siguiente parte de código muestra al dendograma.

```
print('Dendograma entre documentos')
linkage_matrix = ward(dist)
dendrogram(linkage_matrix, orientation="right", labels=titles)
plt.tight_layout()
plt.show()


Dendograma entre documentos
```



Este gráfico describe mejor la forma en que se agrupan los documentos. Los más parecidos son las parejas pg48783 y 49280-0 con una distancia de 0.03, el otro par con la misma distancia es pg61189 y pg50726. En general se forman tres grupos y los colores ayudan a definirlos.
Como antes, la obra que es menos parecida a las demás es pg60464.

Hasta aquí:

1. Haz obtenido datos a partir de obras literarias.
2. Realizaste un preprocesamiento de la información.
3. Calculaste datos descriptivos para identificar diferentes obras con base en un coeficiente de similitud, además de mostrar la similitud de forma gráfica.

En resumen, ya iniciaste el análisis de información realizando minería de datos por lo que cada vez estás más cerca de la ciencia de datos.

CAPÍTULO 14

Computación Cognitiva

14.1 Introducción

De acuerdo a Wikipedia, no existe una definición ampliamente aceptada de computación cognitiva. Sin embargo, el término computación cognitiva (CC) hace referencia a nuevo hardware y/o software que imita el funcionamiento del cerebro humano y ayuda a mejorar la toma de decisiones. En este sentido, CC es un nuevo tipo de computación que tiene como objetivo elaborar modelos más precisos de la forma en que el cerebro/mente humana percibe, razona y responde a los estímulos.

Algunas características que pueden tener los sistemas cognitivos son:

- Aprender a medida que cambia la información y a medida que evolucionan las metas y los requisitos.
- Interactuar fácilmente con los usuarios para que esos usuarios puedan definir sus necesidades cómodamente.
- Ayudar a definir un problema haciendo preguntas o encontrando fuentes de entrada adicionales si el enunciado del problema es ambiguo o incompleto.
- Comprender, identificar y extraer elementos contextuales como el significado, la sintaxis, la hora, la ubicación, el dominio apropiado, las regulaciones, el perfil del usuario, el proceso, la tarea y el objetivo.

Casos de uso:

- Reconocimiento de voz
- Análisis de los sentimientos
- Detección de rostro
- Evaluación de riesgos
- Detección de fraudes
- Recomendaciones de comportamiento

14.2 Servicios de Watson de IBM

Watson es una supercomputadora de IBM. Esta computadora tiene una capacidad de cálculo de 80 teraflops, es decir, 80 × 1012 operaciones de punto flotante por segundo. Esta gran capacidad de cálculo permite analizar gran cantidad de información y utilizar algoritmos complejos de inteligencia artificial (IA) y procedimientos como procesamiento de lenguaje natural y minería de datos.

Watson de IBM es una plataforma de cómputo cognitivo basado en la nube y que ha sido utilizada en diferentes problemas del mundo real. Los sistemas de cómputo cognitivo simulan el reconocimiento depatrones y la capacidad del cerebro humano para tomar de decisiones, además de aprender la forma en que el cerebro mismo consume aún más información.

Watson ofrece un conjunto de técnicas que puedes incorporar en tus aplicaciones. En este capítulo utilizarás técnicas como traducción, y la conversión voz a texto y texto a voz.

Voz a texto. Este servicio de Watson, convierte archivos de audio y los transcribe a texto. El servicio puede distinguir entre múltiples voces. Este servicio lo puedes utilizar para implementar aplicaciones controladas por voz, transcribir audio en vivo, entre otras aplicaciones. En la plataforma de Watson puedes utilizar el demo de prueba, para este servicio:

https://speech-to-text-demo.ng.bluemix.net/

Texto a voz. Este servicio sintetiza voz de texto. Puedes utilizar de síntesis de voz con marcas de lenguaje (Speech Synthesis Markup Language, SSML) para embeber instrucciones en el control de texto como inflexión, cadencia, pitch, entre otras opciones. Hasta hoy, este servicio soporta la síntesis en idiomas como inglés, francés, español, alemán, italiano portugués y japonés. El demo de este servicio está en la dirección:

https://www.ibm.com/demos/live/tts-demo/self-service/home

Traductor. El servicio de traducción tiene dos componentes:

- Traducción entre lenguajes
- Identifica el idioma del texto entre casi 60 lenguajes diferentes

Para utilizar el demo de Watson:

https://www.ibm.com/demos/live/watson-language-translator/self-service
Para inscribirse en el cloud services de Watson, entra en esta dirección:

https://cloud.ibm.com/registration?target=%2Fdeveloper%2Fwatson%2Fdashboard

Recibirás un email con la información necesaria para confirmar el registro de tu cuenta. Después de hacerlo, ya puedes ingresar al tablero de Watson en:

https://cloud.ibm.com/developer/watson/dashboard

Aquí podrás ver el menú de opciones de Watson:

- Kits de inicio
- Servicios de Watson
- Recursos para desarrolladores
- Aplicaciones

IBM proporciona un kit para desarrolladores con Python (software development kit, SDK). Este kit es un módulo que contiene las clases que te permitirán interactuar con los servicios de Watson. Puedes crear objetos específicos para servicios específicos.

Para instalar SDK, abre el Prompt de Anaconda y ejecuta el siguiente comando

```
pip install -upgrade watson-developer-cloud
```

Otros módulos adicionales son:

```
pip install pyaudio
```

```
pip install pydub
```

A continuación, revisaremos un ejemplo sencillo en la forma de aplicar los servicios de Watson. Pero antes de analizar ver el código:

a. Ve a la dirección

   https://console.bluemix.net/atalog/services/speech-to-text

   da click en el botón crear, en la parte inferior derecha de la página. Esto, automáticamente genera un clave API, que te permitirá utilizar el servicio Voz a Texto de Watson.

b. Obtén tus credenciales. Para ver la clave API, da click en gestionar, en la parte superior izquierda. Ahí podrás ver la clave API, con un ícono de copiar ya listo, selecciónalo y pégalo en la variable `voz_a_texto_key` del archivo `claves.py`.

c. Ahora, ve a la dirección

   https://console.bluemix.net/atalog/services/text-to-speech

   da click en el botón crear, en la parte inferior derecha de la página. Esto, automáticamente genera un clave API, que te permitirá utilizar el servicio Texto a Voz de Watson.

d. Obtén tus credenciales. Para ver la clave API, da click en gestionar, en la parte superior izquierda. Ahí podrás ver la clave API, con un ícono de copiar ya listo, selecciónalo y pégalo en la variable text_to_speech_key del archivo claves.py.

e. Finalmente, ve a la dirección

   https://console.bluemix.net/atalog/services/language-traslator

da click en el botón crear, en la parte inferior derecha de la página. Esto, automáticamente genera un clave API, que te permitirá utilizar el servicio de traducción de Watson.

f. Obtén tus credenciales. Para ver la clave API, da click en gestionar, en la parte superior izquierda. Ahí podrás ver la clave API, con un ícono de copiar ya listo, selecciónalo y pégalo en la variable `translate_key` del archivo `claves.py`.

Una vez que ya haz añadido tus credenciales al script claves.py, tal vez sea conveniente que analices la forma en que trabaja la aplicación.

14.3 Ejemplo de un traductor básico

Supongamos que viajas a los Estados Unidos, pero no eres exactamente un experto en hablar inglés. Es más, tienes alguna especie de pánico escénico para hacerlo. Por lo que necesitas de una aplicación que pueda auxiliarte a solicitar servicios básicos, como preguntar donde puedes encontrar un baño, una gasolinera, un banco o comida. Esta aplicación debe ser capaz de trasladar voz en español y traducir en voz a inglés. El otro interlocutor debe escuchar el audio en inglés, responderte en el mismo idioma, pero la aplicación lo llevará al español

```
1  #Traductor_ejemplo.py
2
3  """Traductorbásicoconlosserviciosdewatson."""
4
5  fromwatson_developer_cloudimportSpeechToTextV1
6  fromwatson_developer_cloudimportLanguageTranslatorV3
7  fromwatson_developer_cloudimportTextToSpeechV1

8  import keys # contiene las API keys de Watson
9  import pyaudio # permite acceder a los controles del mi
   crófono 10 import pydub # para cagar archivos WAV
11 import pydub.playback # para reproducir archivos WAV
12 import wave # para guardar archivos WAV
```

Las primeras 3 líneas de código son los módulos para el procesamiento de audio de los servicios de Watson. En la línea 8 está incluido el archivo donde se encuentran tus credenciales, y los siguientes 4 son los módulos de Python que permiten cargar, leer, reproducir y grabar archivos de audio con extensión WAV. Este tipo de archivos fue desarrollado por Microsoft e IBM para almacenar sonido en archivos.
Esta aplicación realiza 10 pasos básicos.

1. Solicita el registro, en español, de un archivo de audio. Inicialmente la aplicación muestra:

   Presiona Enter y haz tu pregunta en español
   Prepara tu pregunta. Cuando presionas Enter, se muestra
   Grabando 5 segundos de audio
   Realiza tu pregunta, por ejemplo, ¿dónde puedo encontrar el banco más cercano? Después de 5 segundos, la aplicación muestra:

```
Grabación completa

13 deftraductor_basico():
14 """Ejemplo de interacción con los servicios de Watson."""
15
16 # Paso 1: Solicita una pregunta en español para grabarla
17 # en un archivo de audio
18 input('Presiona Enter, y haz tu pregunta en español')
19 record_audio('preg_espannol.wav')
20
```

2. La aplicación interactúa con el servicio Voz a Texto de Watson para transcribir el archivo de audio a texto. Al hacerlo, muestra

```
Pregunta en español: dónde puedo encontrar el banco más
cercano

21 # Step 2: Transcribe la voz en español a texto en español
22 texto_espannol = voz_a_texto(file_name = 'preg_espannol.wav',
23 model_id = 'es-ES_SofiaVoice')
24 print('Pregunta en español:', texto_espannol)
25
```

3. Ahora, la aplicación utiliza el servicio de traducción, para traducir del idioma español al idioma inglés. El resultado es el texto traducido

```
English: where is the closest bank

26 # Step 3: Traducción de texto en español a texto en inglés
27 texto_ingles = translate(text_to_translate = texto_es-
pannol, model='es-en')
28 print('Pregunta en inglés:', texto_ingles)
29
```

4. Paso 4. El texto es llevado al servicio Texto a Voz para convertir el texto en un archivo de audio.

```
30 # Paso 4: Sintetiza el texto en inglés, en voz en inglés
31
32 text_to_speech(text_to_speak = texto_ingles, voice_to_
use='es-US_ BroadbandModel',
33 file_name = 'preg_ingles.wav')
```

5. Se escucha el audio resultante en inglés.

```
34 # Paso 5: Reproduce la pregunta en inglés
35 play_audio(file_name='preg_ingles.wav')
36
```

6. En el prompt se muestra
Presiona Enter, y dime con tu voz la respuesta
Al presionar Enter, cuando lo haces se muestra

```
Grabando 5 segundos de audio
```

En esta parte, el otro interlocutor debe decir la respuesta en inglés. Por ejemplo: “The closest bathroom is inside that restaurant”. El archivo de respuesta está disponible en la misma carpeta donde se encuentra el archivo, ejemplo.py. Así que debes estar listo para tocar el archivo de audio cuando pida la repuesta.

Después de 5 segundos en el Prompt aparece

```
Reocording is complete

# Paso 6: Solicita la respuesta en inglés para grabarla en un archivo de audio
input('Presiona Enter, y da tu respuesta en inglés')
record_audio('resp_ingles.wav')

```

7. El programa interactúa con el servicio Voz a Texto para transcribir el archivo de audio a texto y muestra el resultado textual en inglés.

```
Respuesta en inglés: The closest bathroom is inside that restaurant

# Paso 7: Transcribe la voz en inglés, en texto en inglés
texto_r_ingles = voz_a_texto(file_name='resp_ingles.wav',
model_id='en-US_BroadbandModel')
print('Respuesta en inglés:', texto_r_ingles)

```

8. Ahora se emplea el servicio de traducción de inglés a español.

```
Respuesta en español: El baño más cercano está dentro del ese restaurante

# Paso 8: Traducción de texto en inglés a texto en español
resp_texto_esp = translate(text_to_translate= texto_r_ingles, model='en-es')
print('Respuesta en español:', resp_texto_esp)

```

9. Paso 9. Ahora, la aplicación convierte el texto en español a un archivo de audio.

```
# Paso 9: Sintetiza el texto en español, en voz en español
text_to_speech(text_to_speak=resp_texto_esp,voice_to_use='es-ES_SofiaVoice',
file_name='resp_espannol.wav')
```

10. El audio se reproduce con la respuesta en español.

Ahora es momento de analizar el código que realiza todas estas tareas. Esto es lo ocurre detrás del telón:

```
# Paso 10: Reproduce la respuesta en español
play_audio(file_name = 'resp_espannol.wav')


defvoz_a_texto(file_name,model_id):
# Uso de key Speech to Text client de Watson
stt = SpeechToTextV1(iam_apikey=keys.voz_a_texto_key)

# Para abrir los archivos de audio
with open(file_name, 'rb') as audio_file:

# Pasa el archivo de audio a Watson para la transcripción
result = stt.recognize(audio = audio_file,
content_type ='audio/wav', model = model_id).get_result()

# Proporciona una lista de resultados. Puede contender resultados
# parciales o finales
results_list = result['results']

# Los resultados dependen del método de reconocimiento empleado.
# El código únicamente solicita resultados finales, por lo
# que esta lista debe contener un solo elemento

speech_recognition_result = results_list[0]
# Proporciona una lista de alternativas para las transcripciones.
alternatives_list = speech_recognition_result['alternatives']

# La lista puede contener múltiples alternativas.
# Las transcripciones dependen de los argumentos del método de reconocimiento.
# El código no pregunta por diferentes alternativas, por lo que esta
# lista debe contener un solo elemento
first_alternative = alternatives_list[0]

#Obtiene la transciption-key de Watson para hacer la trans
```

```
cripción del audio
transcript = first_alternative['transcript']

# Regresa la transcripción del audio
return transcript

deftranslate(text_to_translate,model):
# Crea el Translator client de Watson
language_translator = LanguageTranslatorV3(version = '2018-05-31',
iam_apikey=keys.translate_key)

 #Realiza la traducción
translated_text = language_translator.translate(
text=text_to_translate, model_id=model).get_result()

#Proporciona una lista de traducciones
translations_list = translated_text['translations']
# La lista puede contener múltiples cadenas
# El código solicita únicamente una cadena, por lo que
# esta lista debe contener  un solo elemento
first_translation = translations_list[0]

#Obtiene la translation-key de Watson para hacer la traducción
translation = first_translation['translation']

# Regresa la traducción
return translation

deftext_to_speech(text_to_speak,voice_to_use,file_name):
# Crea el Text to Speech client de Watson
tts = TextToSpeechV1(iam_apikey=keys.text_to_speech_key)


# Abre y escribe el archivo de audio
with open(file_name, 'wb') as audio_file:
audio_file.write(tts.synthesize(text_to_speak,
accept = 'audio/wav', voice=voice_to_use).get_result().content)

def record_audio(file_name):
# Utiliza pyaudio para grabar 5 segundos de unido en formato WAV
FRAME_RATE = 44100 # Numero de tramas por segundo
CHUNK = 1024 # número de tramas leídos
FORMAT = pyaudio.paInt16 # cada trama es un entero de 16
bits (2 bytes
CHANNELS = 2 # 2 muestra por trama
SECONDS = 5 # Tiempo de grabación

recorder = pyaudio.PyAudio() # abre/cierra la grabación de audio

# Configuración para la grabación de preguntas y respuestas
audio_stream = recorder.open(format=FORMAT, channels=CHANNELS,
rate=FRAME_RATE, input=True, frames_per_buffer=CHUNK)
audio_frames = [] # stores raw bytes of mic input
print('Grabando 5 segundos de audio')

for i in range(0, int(FRAME_RATE * SECONDS / CHUNK)):
audio_frames.append(audio_stream.read(CHUNK))

print('Grabación completa')
audio_stream.stop_stream() # Detiene la grabación
audio_stream.close()
recorder.terminate() # librería de recursos de PyAudio

# guarda las tramas de audio en un archivo WAV
with wave.open(file_name, 'wb') as output_file:
output_file.setnchannels(CHANNELS)
output_file.setsampwidth(recorder.get_sample_size(FORMAT))
output_file.setframerate(FRAME_RATE)
output_file.writeframes(b''.join(audio_frames))

defplay_audio(file_name):
#Utiliza el módulo pydub (pip install pydub) para reproducir
archivos WAV
sound = pydub.AudioSegment.from_wav(file_name)
pydub.playback.play(sound)

if__name__=='__main__':
traductor_basico()
```

CAPÍTULO 15

Introducción a Machine Learning

15. 1 Introducción

Machine Learning es un tema grande y complejo que puede emplearse en muchos problemas pero que se caracteriza por plantear soluciones sutiles, es decir, en lugar de utilizar la experiencia del programador para resolver alguna situación específica para tales problemas, la programación está orientada a aprender de los datos.

En este capítulo se presentan diversos ejemplos para construir modelos de Machine learning que puedes utilizar para realizar predicciones con gran precisión.

15.2 Tipos de Machine Learning

Existen básicamente dos tipos de Máquinas de Aprendizaje (Machine Learning)

- Machine Learning Supervisado. Si los datos para trabajar tienen etiquetas.

  Un ejemplo de este tipo de aprendizaje es la identificación de personas mediante el reconocimiento facial. El algoritmo se entrena con un conjunto de fotos de los integrantes de una empresa. Al momento de ingresar a las instalaciones, el algoritmo reconoce a los miembros de la empresa y también a aquellos que no lo son.

  El aprendizaje supervisado emplea dos técnicas específicas,

  1. La clasificación en donde se aplican técnicas de análisis multivariado como el análisis de cluster o análisis de discriminación
  2. El pronóstico, donde se aplican técnicas de regresión. Y que puede ser regresión simple, múltiple o bayesiana.

- Machine Learning No Supervisado. Si los datos para trabajar no tienen etiquetas o si se pretende realizar una reducción de variables.

  Este tipo de aprendizaje lo pueden utilizar los reclutas que buscan candidatos para postularlos en algún puesto específico. El algoritmo debe realizar sugerencias de aquellas personas cuyo currículum se adapta más a un perfil específico. En términos académicos, el algoritmo podría ser útil para identificar estudiantes con características comunes que puedan tener problemas de reprobación o deserción.

  El aprendizaje no supervisado emplea también técnicas del análisis de clúster y de reducción de dimensiones como análisis de componentes principales entre otros métodos.

La secuencia de un estudio típico de Ciencia de Datos:

- Cargar la base de datos
- Exploración básica de datos
- Transformación de datos
- Separación de datos para entrenamiento y prueba
- Creación de un modelo
- Entrenamiento y prueba del modelo
- Ajuste y evaluación del modelo
- Realizar predicciones

15.3 Machine Learning Supervisado. Clasificación y Pronóstico

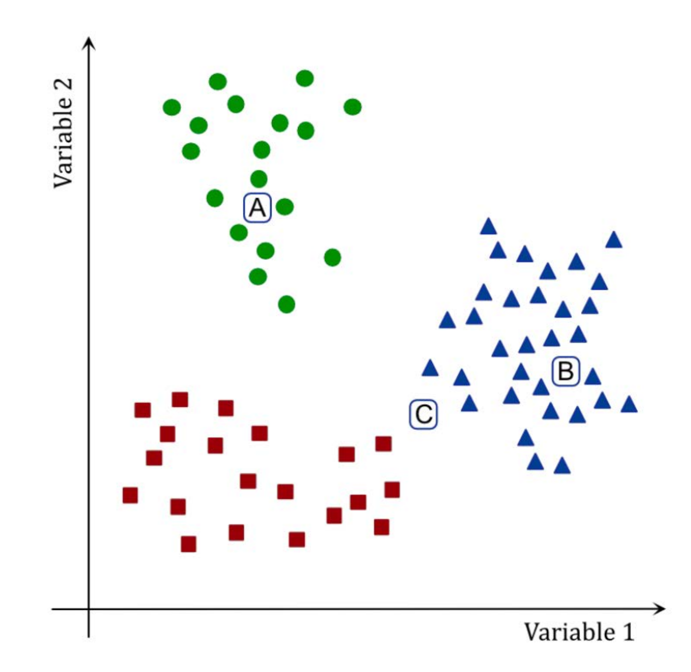
Para ejemplificar el problema de la clasificación, puedes utilizar la base de datos Digits incluido en Scikit-Learn.

Scikit-Learn (o sklearn) es una librería de Python para Machine Learning. Las técnicas que emplea están encapsuladas, por lo que no es fácil acceder a la intrincada complejidad de los algoritmos que utiliza. Con sklearn es posible entrenar y probar el modelo.

La base de datos Digits contiene 1797 imágenes de los números dígitos escritos a mano. El objetivo es predecir el dígito de una imagen específica. Como hay 10 dígitos, este es un problema de clasificación múltiple.

En este caso es posible utilizar aprendizaje supervisado, ya que cada imagen tiene una etiqueta. Para identificar los diferentes números, el algoritmo de clasificación más utilizado es el del vecino más cercano, (k- nearest neighbors, k-NN)

Una forma gráfica de explicar cómo funciona el algoritmo k-NN es la siguiente:



Se requiere predecir a que clase se deben asignar las muestras A, B y C. Si seleccionamos como parámetro $k = 4$, el algoritmo k-NN realizará la asignación utilizando los cuatro vecinos más cercanos de cada muestra. Entonces:

- Los cuatro vecinos más cercanos a la muestra A son todos círculos verdes. Así, se estima que A pertenece a la clase círculos verdes.
- Los cuatro vecinos más cercanos a la muestra B son todos triángulos azules. Así, se estima que B pertenece a la clase triángulos azules.
- En el caso de la muestra C, no es evidente la clase a la cual hay que asignarla. Pero siguiendo el algoritmo k-NN, de los 4 vecinos más cercanos, 3 son triángulos azules. Por lo que la muestra C debe asignarse a la clase triángulos azules.

Iniciaremos la tarea de clasificar números dígitos con la base de datos de Scikit-Learn.

Inicialmente incluiremos las librerías necesarias, en este caso

```
[1]: %matplotlib inline
     import matplotlib.pyplot as plt
[2]: from sklearn import datasets, svm, metrics
[3]: from sklearn.model_selection import train_test_split
```

En la primera línea, el comando %matplotlib inline es necesario para que los resultados aparezcan en el área de trabajo de Jupyter, si no se incluye, las gráficas aparecerán en otras ventanas. Como en otras ocasiones, pyplot, proporciona una caja de opciones para generar gráficos personalizados. Las siguientes dos líneas invocan librerías de sklearn.

- El módulo datasets contiene una gran cantidad de bases de datos, los de nuestro interés son los números dígitos escritos a mano.

- El módulo svm (Vector Support Machines) o máquina de soporte vectorial, es un algoritmo de clasificación inicialmente diseñado para clasificaciones binarias, pero que se ha desarrollado para implementarse en otros métodos de y de regresión.

- Em módulo metrics, proporciona una serie de métricas que permiten calcular distancias (similaridad) entre dos objetos. Esto es, si la distancia medida, a través de las variables de los objetos, es pequeña, entonces los objetos son parecidos (similares), en otro caso no son similares.

- La función model_selection divide al conjunto de datos en dos: los de entrenamiento y los de prueba, esta comparación es una estrategia que permite identificar la precisión del algoritmo.

15.4 Cargar la base de datos

Para cargar la base de datos para trabajar

```
[4]: digits=datasets.load_digits()
```

La función load_digits() del módulo sklearn.datasets regresa un objeto de scikit-learn que contiene la información de la base de datos Digits. Si quieres conocer esta descripción, puedes escribir print(digits.DESCR).

La base de datos contiene 1797 muestras (imágenes de los dígitos), cada una con 64 valores en un rango de 0 a 16 y que representan la intensidad de un pixel. Con Matplotlib, es posible visualizar las intensidades en escalas de grises, desde le blanco (0) hasta el negro (16).



El arreglo objetivo (target) contiene las etiquetas de las imágenes. El arreglo es llamado target, debido a que compararás un método de identificación con las etiquetas de las imágenes que sumiremos correctas. Esto es, si el método dice que la imagen es 8 y coincide con la etiqueta 8, la clasificación será correcta, pero, puede ocurrir que no lo sea.

Para ver las etiquetas de algunas muestras del conjunto de datos.

```
[5]: digits.target[::100]
[5]: array([0, 4, 1, 7, 4, 8, 2, 2, 4, 4, 1, 9, 7, 3, 2, 1, 2, 5])
```

Para ver el número de valores por cada muestra, puedes utilizar el atributo shape.

```
[6]: digits.data.shape
[6]: (1797, 64)
```


Esto indica el arreglo digits tiene 1797 filas, donde cada fila representa una muestra, y 64 columnas, que representan los valores de cada muestra.

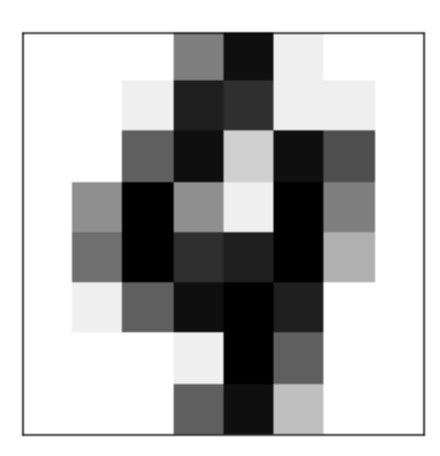
Cada imagen representa un arreglo de dimensiones 8x8, por ejemplo, el arreglo que representa a la imagen con índice 14 es:

```
[7]: digits.images[14]
[7]: array([[ 0., 0.,  0.,  8., 15.,  1.,  0.,  0.],
            [ 0., 0.,  1., 14., 13.,  1.,  1.,  0.],
            [ 0., 0., 10., 15.,  3., 15., 11.,  0.],
            [ 0., 7., 16.,  7.,  1., 16.,  8.,  0.],
            [ 0., 9., 16., 13., 14., 16.,  5.,  0.],
            [ 0., 1., 10., 15., 16., 14.,  0.,  0.],
            [ 0., 0.,  0.,  1., 16., 10.,  0.,  0.],
            [ 0., 0.,  0., 10., 15.,  4.,  0.,  0.]])
```

Para mostrar a la imagen con índice 14

```
[8]: axes=plt.subplot()
[9]: image=plt.imshow(digits.images[14],cmap=plt.cm.gray_r)
[10]: axes.set_xticks([])
[11]: axes.set_yticks([])
[11]: []
```

Este número parece un 9, para verificar si lo es:



```
[12]: digits.target[14]
[12]: 4
```

Sorprendentemente, el dígito que representa es un 4 y no un 9 como lo había supuesto.

15.5 Transformación de datos

Para prepara los datos, necesitas conjuntos de datos donde las filas representen muestras (o individuos) y columnas que indiquen los valores de las variables. Esto es lo que requieren las máquinas de aprendizaje de Scikit-learn.

Si observas el arreglo del snippet [7] regresa un arreglo de 8x8, sin embargo digits.data[14] acomoda la información en una sola fila

```
[13]: digits.data[14]
[13]: array([ 0.,  0.,  0.,  8., 15.,  1.,  0.,  0.,  0.,
              0.,  1., 14., 13.,  1.,  1.,  0.,  0.,  0.,
             10., 15.,  3., 15., 11.,  0.,  0.,  7., 16.,
              7.,  1., 16.,  8.,  0.,  0.,  9., 16., 13.,
             14., 16.,  5.,  0.,  0.,  1., 10., 15., 16.,
             14.,  0.,  0.,  0.,  0.,  0.,  1.,  6., 10.,
              0.,  0.,  0.,  0.,  0., 10., 15.,  4.,  0.,
              0.])
```

De dimensiones 1x64.

15.6 Exploración básica de datos

Esto consiste en familiarizarte con el tipo de datos que estás trabajando y esto se realiza explorando un poco la información. En este caso, puedes visualizar la información de la base de datos

```
[14]: _, axes = plt.subplots(nrows=3, ncols=5, figsize=(10,12))
for ax, image, label in zip(axes.ravel(), digits.images, digits.
target):
   ax.set_axis_off()
   ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
   ax.set_title('Etiqueta: %i' % label)
plt.tight_layout()
plt.savefig("out.png",bbox_inches='tight',pad_inches=0)
```

Etiqueta: 0 Etiqueta: 1 Etiqueta: 2 Etiqueta: 3 Etiqueta: 4
Etiqueta: 5 Etiqueta: 6 Etiqueta: 7 Etiqueta: 8 Etiqueta: 9
Etiqueta: 0 Etiqueta: 1 Etiqueta: 2 Etiqueta: 3 Etiqueta: 4

En el snippet[14], el guión bajo (underscore) tiene 5 funciones diferentes. Una de ellas es la de ignorar ciertos valores, es decir, la función `plt.subplots` puede regresar diversos atributos, de los cuales, el único que es de interés son `axes`.

`plt.subplot`, construye un arreglo con las imágenes que quieres cargar. En este ejemplo, el gráfico generado tiene 3 filas y 5 columnas para 15 imágenes posibles, y tendrá un tamaño de 10x12 pulgadas.

En cada iteración del ciclo `for`, se descomprime una tupla en tres variables: el objeto axes (`ax`), la imagen (`image`) y la etiqueta de la imagen (`label`)

Se invoca al objeto `Axes` del método (imshow) para mostrar una imagen. El keyword: `cmap=plt.cm.gray_r` determina el mapa de colores (17 en la escala de grises) para mostrar la imagen.

Los títulos en cada imagen se definen a través de `ax.set_title`, con título Etiqueta y un entero que indica el `label` de la imagen.

La tarea de `plt.tight_layout()` es la de remover el espacio en blanco, en los extremos superiores, inferiores y a la derecha e izquierda del gráfico.

Por último, `plt.savefig` guarda el gráfico con el nombre de out, en formato png, en la misma carpeta donde se ejecuta el programa.

15.7 Separación de datos para Entrenamiento y Prueba

De un solo conjunto de datos, haremos dos partes: una parte que servirá de entrenamiento y otro que será parte de las pruebas del método de reconocimiento.

La función `train_test_split` del módulo sklearn.model_selection realiza una especie de división en dos muestras y aleatorización. Esto permite quitar el sesgo al momento de seleccionar las muestras y les da características similares.

La función `train_test_split` regresa una tupla de 4 elementos. Los primeros 2 son las muestras divididas en conjuntos de entrenamiento y prueba. Los otros corresponden a los valores objetivo (targets), también de entrenamiento y de prueba.

Por convención, utilizaremos la X mayúscula para representar a las muestras y la y minúscula para representar a los valores objetivo.

```
[15]:  X_entrena, X_prueba, y_entrena, y_prueba = train_
test_split(digits.data, digits.target, random_state=1)
```

Con el objetivo de verificar la reproductibilidad de los resultados, la semilla para generar números pseudoaleatorios la dejaremos fija. Esto es con random_state=1 podrás comprobar que se obtienen los mismos resultados.

Por default, `train_test_split` asigna el 75% de los datos para el entrenamiento y 25% para las pruebas

```
[16]: X_entrena.shape
[16]: (1374, 64)
[17]: X_prueba.shape
[17]: (450, 64)
```

Para especificar diferentes porcentajes para realizar la división de los datos, es posible añadir un argumento a la función `train_test_split`, por ejemplo, si haz decidido dejar el 40% de los datos para realizar pruebas, entonces, el snippet [] se modifica como

```
X_entrena, X_prueba, y_entrena, y_prueba = train_test_split(digits.
                data, digits.target, random_state=1, test_size=0.40)
```

Dejando así 60% de los datos para realizar el entrenamiento.

15.8 Creación del modelo

El estimador `KNeighborsClassifier` implementa el algoritmo del vecino más cercano. Primero, crearemos un objeto estimador `KNeighborsClassifier`.

```
[18]: from sklearn.neighbors import KNeighborsClassifier
[19]: knn = KNeighborsClassifier()
```

15.9 Entrenamiento y prueba del modelo

Invocaremos al método fit de `KNeighborsClassifier` que carga al conjunto de imágenes (X_entrena) y de targets (y_entrena) de entrenamiento en el estimador. Esta parte es la que caracteriza al machine learning supervisado.

```
[20]: knn.fit(X=X_entrena, y=y_entrena)
[20]: KNeighborsClassifier()
```

En este caso se han dejado los parámetros por default de la fusión knn.fit, los cuales son:

```
algorithm='auto'       metric_params=none     p=2
leaf_size=30           n_jobs=none            weights='uniform'
metric='minkowski'     n_neighbors=5
```

Si requieres conocer más información de `KNeighborsClassifier`, puedes ingresar al sitio:

https://scikit-learn.org/stable

Y en el buscador escribir sklearn.neighbors.KNeighborsClassifier

La documentación de este método, está disponible en

https://scikit-learn.org/stable/modules/neighbors.html#classification

15.10 Pronóstico de clases de dígitos

Ahora es posible utilizar el conjunto de pruebas para realizar pronósticos. El estimador del método predict con X_prueba regresa un array que contiene la clase pronosticada para cada imagen de prueba.

```
[21]: pronostico=knn.predict(X=X_prueba)
[22]: esperado=y_prueba
[23]: pronostico[:20]
[23]: array([6, 0, 5, 3, 2, 9, 0, 4, 1, 0, 1, 8, 2, 5, 2, 8, 1, 8, 9, 1])
[24]: esperado[:20]
[24]: array([6, 0, 5, 9, 2, 9, 0, 4, 1, 0, 1, 8, 2, 5, 2, 8, 1, 8, 9, 1])
```

Observa que se presentan las primeras 20 muestras. Al realizar un contraste rápido puedes observar que la muestra con índice 3, fue asignada a la clase de dígitos 3, cuando esperábamos que fuera asignada a la clase de dígitos 9.

Utilizaremos una lista de compresión para localizar todos los pronósticos erróneos en todas las pruebas.

```
[25]: errores=[(p,e) for (p,e) in zip(pronostico,esperado) if p!=e]
[26]: errores
[26]: [(3, 9),
       (3, 9),
       (5, 9),
       (7, 3),
       (1, 9),
       (1, 8),
       (9, 3),
       (3, 8),
       (1, 8),
       (3, 8)]
```

En las tuplas, el primer valor p, es el valor del pronóstico, y el segundo es el de la etiqueta (objetivo). La tupla se muestra cuando estos valores son diferentes. En este ejemplo, solo 10 de 450 valores fueron incorrectos, por lo que el porcentaje de precisión del estimador es de 97.77%, este es un buen porcentaje, ya que incluso, se han dejado los valores por default del estimador.

15.11 Ajuste y evaluación del modelo

Cada estimador tiene un método de evaluación (score) que regresa un indicador de que tan bien trabaja el estimador para los datos de prueba

```
[27]: print(f'{knn.score(X_prueba,y_prueba):0.2%}')
[27]: 97.78%
```

El método con su valor por default k=5, obtiene un 97.78% de precisión en los pronósticos.

Otra forma de revisar la precisión de la clasificación es a través de la matriz de confusión, la cual muestra los valores de los pronósticos correctos e incorrectos para cada clase.

```
[28]: from sklearn.metrics import confusion_matrix
[29]: mat_confusion=confusion_matrix(y_true=esperado,y_pred=pronostico)
[30]: mat_confusion
[30]:array([[52,  0,  0,  0,  0,  0,  0,  0,  0,  0],
           [ 0, 48,  0,  0,  0,  0,  0,  0,  0,  0],
           [ 0,  0, 48,  0,  0,  0,  0,  0,  0,  0],
           [ 0,  0,  0, 41,  0,  0,  0,  1,  0,  1],
           [ 0,  0,  0,  0, 47,  0,  0,  0,  0,  0],
           [ 0,  0,  0,  0,  0, 38,  0,  0,  0,  0],
           [ 0,  0,  0,  0,  0,  0, 38,  0,  0,  0],
           [ 0,  0,  0,  0,  0,  0,  0, 41,  0,  0],
           [ 0,  2,  0,  2,  0,  0,  0,  0, 46,  0],
           [ 0,  1,  0,  2,  0,  1,  0,  0,  0, 41]], dtype=int64)
```

Los valores en la diagonal principal representan los pronósticos correctos, los valores no-cero fuera de la diagonal indican pronósticos erróneos.

Cada fila representa una clase distinta (dígitos de 0 a 9). Las columnas dentro de una fila, indica cuántas pruebas fueron clasificadas en cada una de las distintas clases.

Por ejemplo, la fila 0:

```
[52,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

Representa la clase del dígito 0. Las columnas representan los 10 posibles objetivos (targets) de 0 a 9. De acuerdo a la fila 0, 52 muestras fueron clasificadas en la clase 0 y ninguna muestra fue asignada a otra clase de manera incorrecta. Por lo que el 100% de los 0's fueron correctamente clasificados.

Otro ejemplo, la fila 9:

```
[ 0,  1,  0,  2,  0,  1,  0,  0,  0, 41]
```

Representa la clase del dígito 9. De acuerdo a la fila 9, 41 muestras fueron clasificadas en la clase 9, pero:

- El 1 de la columna con índice 1, indica que un 9 fue incorrectamente clasificado como 1.
- El 2 de la columna con índice 3, indica que dos 9's fueron incorrectamente clasificados como 3.
- El 1 de la columna con índice 5, indica que un 9 fue incorrectamente clasificado como 5.

Por lo que el algoritmo hizo un pronóstico correcto de 91.11% (41 de 45) de 9's.

De acuerdo a la matriz de confusión, del método de estimación, y con los parámetros seleccionados en el estimador y en las semillas de valores pseudoaleatorios, los números 8 y 9 son los que presentan una aparente mayor dificultad para ser reconocidos.

El módulo `sklearn.metrics` también proporciona la función `classification_report`, la cual produce una tabla de clasificación de métricas basada en los valores esperados y pronosticados

```
[31]: from sklearn.metrics import classification_report
[32]: names=[str(digit) for digit in digits.target_names]
[33]: print(classification_report(esperado,pronostico,-
target_names=names))
[33]:           precision    recall  f1-score   support
          0       1.00      1.00      1.00        52
          1       0.94      1.00      0.97        48
          2       1.00      1.00      1.00        48
          3       0.91      0.95      0.93        43
          4       1.00      1.00      1.00        47
          5       0.97      1.00      0.99        38
          7       0.98      1.00      0.99        41
          8       1.00      0.92      0.96        50
          9       0.98      0.91      0.94        45
   accuracy                           0.98       450
  macro avg       0.98      0.98      0.98       450
weighted avg      0.98      0.98      0.98       450
```

- `precision`, es el número total de pronósticos correctos para cada clase, dividido entre el número total de pronósticos para tal dígito. Por ejemplo, en la columna con índice 3, hay 2's en la fila 8 y 9 indicando que dos 8's y dos 9's fueron clasificados como 3. Y el 41 de la misma columna, indica que 41 imágenes fueron correctamente clasificadas. La precisión para el número 3 es de 41/45 o 0.91.

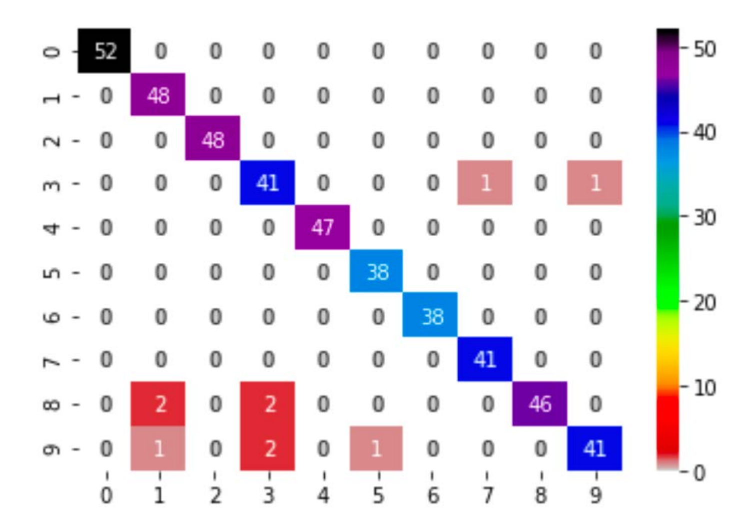
- `recall`, es el número total de pronósticos correctos para cada clase, dividida entre el total del número de muestras que han sido pronosticadas en tal clase. Por ejemplo, la fila 3, hay un 7 y un 9 indicando que dos 3's fueron clasificados en otras clases. El número 41, indica que 41 imágenes fueron correctamente clasificadas, por lo que el recall, para el dígito 3 es 41/43 o 0.95.

- `f1-score`, es el promedio entre precision y recall.

- `support`, es el número de muestras con un valor esperado dado. Por ejemplo, en la fila 3, se esperaban 43 clasificaciones correctas.

Un mapa de calor muestra con colores aquellos valores de mayor magnitud. Las funciones para graficar de seaborn pueden realizarlo automáticamente con datos de dos dimensiones. Convertiremos la matriz de confusión en un DataFrame para graficarlo.

```
[34]: import pandas as pd
[35]: confusion_df = pd.DataFrame(mat_confusion,index =
      range(10),columns=range(10))
[36]: import seaborn as sns
[37]: axes=sns.heatmap(confusion_df,annot=True,cmap=
      'nipy_spectral_r')
```



El argumento `annot=True` de la función `heatmap` muestra una barra de colores a la derecha. El `cmap='nipy_spectral_r'` el argumento que especifica el mapa de colores a utilizar, en este caso `'nipy_spectral_r'`.

15.12 Validación Cruzada K-Fold

La validación cruzada habilita el uso de todos los datos para entrenar y probar, en el sentido de que tan bueno puede ser el modelo para realizar pronósticos, en nuevos datos, mediante la repetición y prueba con diferentes proporciones de conjuntos de datos. La validación cruzada K-fold divide al conjunto de datos en k grupos (folds) de igual tamaño. El algoritmo trabaja así: considera el uso de k=10 grupos, numerados del 1 al 10. Con 10 grupos, realizaremos 10 ciclos de entrenamientos y pruebas, es decir

- Primero, entrenamos con los conjuntos 1 al 9, y probamos con el grupo 10.
- Ahora, entrenamos con los conjuntos 1 al 8, y probamos con el grupo 9.
- Ahora, entrenamos con los conjuntos 1 al 7, y probamos con el grupo 8.

Este ciclo de entrenamiento y prueba continua hasta que cada grupo ha sido utilizado para probar el modelo.

Scikit-learn tiene la clase KFold y la función cross_val_score (ambas en el módulo sklearn.model_selection) para realizar los ciclos de entrenamiento y prueba. Primero crearemos un objeto KFold.

```
[38]: from sklearn.model_selection import KFold
[39]: kfold=KFold(n_splits=10,random_state=7,shuffle=True)
```

Los argumentos son:

- `n_splits=10`, éste es el número de grupos (folds).
- `random_state=11`, es la misma semilla que habíamos utilizado anteriormente.
- `shuffle=True`, que tiene la tarea de aleatorizar los datos antes de separarlos en grupos.

Ahora se utiliza la validación cruzada cross_val_score para entrenar y probar el modelo

```
[40]: from sklearn.model_selection import cross_val_score
[41]: evaluaciones = cross_val_score(estimator=knn,
      X=digits.data, y=digits.target, cv=kfold)
```

Los argumentos son:

- `estimator=knn`, aquí se define el estimador que quieres validar.
- `X=digits.data`, especifica las muestras para entrenamiento y prueba.
- `y=digits.target`, define los pronósticos objetivo para las muestras.
- `cv=kfold`, que especifica el generador de la validación cruzada y que define la forma de dividir las muestras y los targets para entrenamiento y prueba.

```
[42]: evaluaciones
[43]: array([0.95555556, 0.99444444, 0.99444444, 0.98888889, 1.        ,
       0.97222222, 0.98888889, 0.97765363, 0.98882682, 0.98882682])
```

Una vez que se tiene la matriz de precisión de los 10 grupos, es posible evaluar la precisión de todo el modelo, mediante la evaluación promedio de la precisión y la respectiva desviación estándar.

```
[44]: print(f'Precisión promedio:{evaluaciones.mean():0.2%}')
[45]: Precisión promedio:98.50%
[46]: print(f'DesviaciónestándardelaPrecisión:
      {evaluaciones.std():0.2%}')
[47]: Desviación estándar de la Precisión:1.24%
```

En promedio, el modelo tuvo una precisión del 98.5%, que resulta ligeramente mejor del 98% que se había logrado, cuando se realizó un ciclo con 75% de datos de entrenamiento y con 25% de datos de prueba.

Además de todas las bondades que ha mostrado Scikit-learn también permite comparar la precisión con diferentes modelos y determinar a la mejor máquina de aprendizaje.

A continuación, se presenta los estimadores KNeighborsClassifier, SVC y GaussianNB (pero hay más). Los últimos dos no los habíamos utilizado, pero resultan fáciles de implementar.

Para importar las librerías necesarias:

```
[48]: from sklearn.svm import SVC
[49]: from sklearn.naive_bayes import GaussianNB
```

Ahora crearemos los estimadores, justo como lo realizamos con el método KNeighborsClassifier

```
[50]: estimadores={'KNeighborsClassifier':knn,
                   'SVC': SVC(gamma='scale'),
                   'GaussianNB': GaussianNB()}
```

Y ahora ejecutaremos los modelos

```
[51]: for estimator_name, estimator_object in estimadores.items():
          kfold=KFold(n_splits=10,random_state=7,shuffle=True)
          evaluaciones=cross_val_score(estimator=estimator_object,
          X=digits.data,
                            y=digits.target,cv=kfold)
          print(f'{estimator_name:>20}: '+
                f'Precisión promedio={evaluaciones.mean():0.2%} '+
                f'Desviación estándar={evaluaciones.std():0.2%} ')
[51]: KNeighborsClassifier: Precisión promedio=98.50%
      Desviación estándar=1.24%
                       SVC: Precisión promedio=98.83%
      Desviación estándar=0.76%
                GaussianNB: Precisión promedio=83.41%
      Desviación estándar=3.90%
```

En este ciclo las tareas fueron

- Desempacar la clave dentro de estimator_name y value en un estimator_object.
- Crear un objeto KFold que mezcla los datos y genera 10 grupos. Aquí el dejar fijo a la semilla permite que todos los estimadores trabajen bajo las mismas condiciones.
- Evalúa cada uno de los estimator_object utilizando la validación cruzada.
- Al final muestra el nombre del estimador, el promedio y la desviación estándar de las precisiones de los 10 grupos.

Basados en los resultados el estimador SVC es ligeramente mejor que los demás. Aunque todos pueden mejorar las precisiones, si se atienden mejor los parámetros del estimador. `SVC` y `KNeighborsClassifier` presentan resultados casi idénticos, por lo que son candidatos a revisar los parámetros.

15.13 Regresión Lineal Múltiple

En esta sección trabajarás el tópico de análisis de regresión múltiple. Un modelo de regresión lineal múltiple tiene la forma:

$$y=\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n X_n+\varepsilon$$

Donde:

- y, es la var
- iable independiente.
- $x_1, x_2, \ldots, x_n$, son las variables independientes.
- $\beta_0, \beta_1, \ldots, \beta_n$, son los coeficientes del modelo y tienen valores desconocidos.
- ε, es una variable aleatoria desconocida.

Como el modelo presenta valores desconocidos, el propósito inicial de la regresión es el realizar una estimación de los coeficientes $\beta_0, \beta_1, \ldots, \beta_n$.y construir un modelo de la forma.

$$\hat{y}=\hat{\beta}_0+\hat{\beta}_1 x_1+\hat{\beta}_2 x_2+\ldots+\hat{\beta}_n X_n$$

Donde los coeficientes $\hat{\beta}_i$ son estimadores de los coeficientes verdaderos β_i y en consecuencia $\hat{y}$ es solo una aproximación (estimación) del valor y . Observa que en este último modelo no se ha incluido la variable ε. Esto es válido, si suponemos que en promedio el valor de esta variable, es cero.

Aplicaremos las técnicas de regresión a una base de datos de scikit-learn, llamada load_boston.

15.14 Cargar la base de datos

Para cargar la base de datos de sklearn

```
[1]: %matplotlib inline
     from sklearn.datasets import load_boston
```

```
[2]: boston = load_boston()
```

15.15 Exploración básica de datos

Para ver los detalles de las variables involucradas, es posible invocar una descripción de la base de datos:

```
[3]: print(boston.DESCR)
     .. _boston_dataset:
     Boston house prices dataset
     ---------------------------
     **Data Set Characteristics:**

       :Number of Instances: 506
      :Number of Attributes: 13 numeric/categorical predictive. Median
Value (attribute 14) is usually the target.

        :Attribute Information (in order):
          -CRIM per capita crime rate by town
         -ZN proportion of residential land zoned for lots over 25,000
sq.ft.
          -INDUS per capita crime rate by town
          -CHAS Charles River dummy variable (= 1 if tract bounds ri-
ver; 0 otherwise)
         -NOX nitric oxides concentration (parts per 10 million)
         -RM average number of rooms per dwelling
         -AGE proportion of owner-occupied units built prior to 1940
         -DIS weighted distances to five Boston employment
```

```
centres
      -RAD index of accessibility to radial highways
      -TAX full-value property-tax rate per $10,000
      -PTRATIO pupil-teacher ratio by town
      -B 1000(Bk - 0.63)^2 where Bk is the proportion of
blacks by town
      -LSTAT % lower status of the population
      -MEDV Median value of owner-occupied homes in $1000's

   :Missing Attribute Values: None

   :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databa-
ses/housing/

This dataset was taken from the StatLib library which is
maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld,
D.L. 'Hedonic prices and the demand for clean air', J. En-
viron. Economics & Management, vol.5, 81-102, 1978. Used in
Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley,
1980. N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine
learning papers that address regression problems.

.. topic:: References

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identif-
ying Influential Data and Sources of Collinearity', Wiley,
1980. 244-261.
- Quinlan,R. (1993). Combining Instance-Based and Model-Ba-
sed Learning. In Proceedings on the Tenth International
Conference of Machine Learning, 236-243, University of Mas-
sachusetts, Amherst. Morgan Kaufmann.
```

De acuerdo con la información en la descripción, esta base de datos contiene la información de 506 individuos (viviendas) con 13 variables diferentes, de las cuales una sirve como variable de respuesta y es la última. Esta variable está definida como Y, y representa el valor medio de las viviendas ocupadas por sus propietarios, medida en miles de dólares.

La base de datos cargada, no está representada como una mezcla de datos (buch) ya están bien definidas las variables independientes y la dependiente (target).

```
[4]: boston.data.shape
[4]: (506, 13)
[5]: boston.target.shape
[5]: (506, )
```

Las variables independientes son

```
[6]: boston.feature_names
[6]: array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'],
dtype='<U7')
```

Para explorar los datos utilizaremos Pandas

```
[7]:  import pandas as pd
[8]:  pd.set_option('precision', 4)
[9]:  pd.set_option('max_columns', 8)
[10]: pd.set_option('display.width', None)
```

En este caso,

- `'precision'`, es el número máximo de decimales que se mostrarán en los cálculos.
- `'max_columns'`, es el número máximo de columnas que se mostrarán en el DataFrame.
- `'display.width'`, especifica el ancho de caracteres en el Prompt, en este caso no hay ningún límite.

Ahora, construiremos un DataFrame con los datos,

```
[11]:  boston_df  = pd.DataFrame(boston.data, columns=bos-
ton.feature_names)
[12]: boston_df['MEDV'] = pd.Series(boston.target)
```

Observa que en el snippet [11] se definen las variables independientes (`'CRIM', 'ZN', 'IN-DUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'`) y en el siguiente snippet la variable objetivo (target) o que también hemos llamado variable dependiente.

Para ver la información del DataFrame

```
[13]: boston_df.head()
[13]:     CRIM    ZN    INDUS ...  B       LSTAT  MEDV
       0  0.0063  18.0    2.31...  396.90  4.98   24.0
       1  0.0273  0.0     7.07...  396.90  9.14   21.6
       2  0.0273  0.0     7.07...  392.83  4.03   34.7
       3  0.0324  0.0     2.18...  394.63  2.94   33.4
```

```
            4  0.0691  0.0     2.18...  396.90  5.33  36.2
            5 rows × 14 columns
```

Puedes dar una revisión rápida de la información mediante un resumen de estadísticas descriptivas.

```
[14]: boston_df.describe()
[14]:  CRIM        ZN     INDUS ...         B     LSTAT      MEDV
count  506.0000 506.0000 506.0000 ... 506.0000 506.0000 506.0000
 mean    3.6135  11.3636  11.1368 ... 356.6740  12.6531  22.5328
  std    8.6015  23.3225   6.8604 ...  91.2949   7.1411   9.1971
  min    0.0063   0.0000   0.4600 ...   0.3200   1.7300   5.0000
  25%    0.0820   0.0000   5.1900 ... 375.3775   6.9500  17.0250
  50%    0.2565   0.0000   9.6900 ... 391.4400  11.3600  21.2000
  75%    3.6771  12.5000  18.1000 ... 396.2250  16.9559  25.0000
  max   88.9762 100.0000  27.7400 ... 396.9000  37.9700  50.0000
 8 rows x 14 columns
```

Siempre resulta revelador visualizar los datos de las variables independientes vs la variable dependiente. De realizarlo de manera simultánea, estaríamos navegando en un espacio con 14 dimensiones. Lo cierto es que no hay ni idea de que es eso o que estaríamos viendo, ni tampoco hay algoritmos que lo realicen.

Para hacer la visualización más clara graficaremos a las variables por pares, y utilizaremos el método sample de DataFrame, seleccionando aleatoriamente solo el 30% de las 506 muestras, solo con el propósito de mostrar las gráficas y analizar el comportamiento de las variables.

Nuevamente, con el objetivo de hacer este análisis reproducible, queda fija la semilla para realizar la misma selección de datos.

```
[15]: sample_df = boston_df.sample(frac=0.3, random_state=7)
```

Ahora construiremos los diagramas de dispersión de cada una de las variables independientes

`CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT, contra MEDV`

Solo para recordar el significado de estas variables:

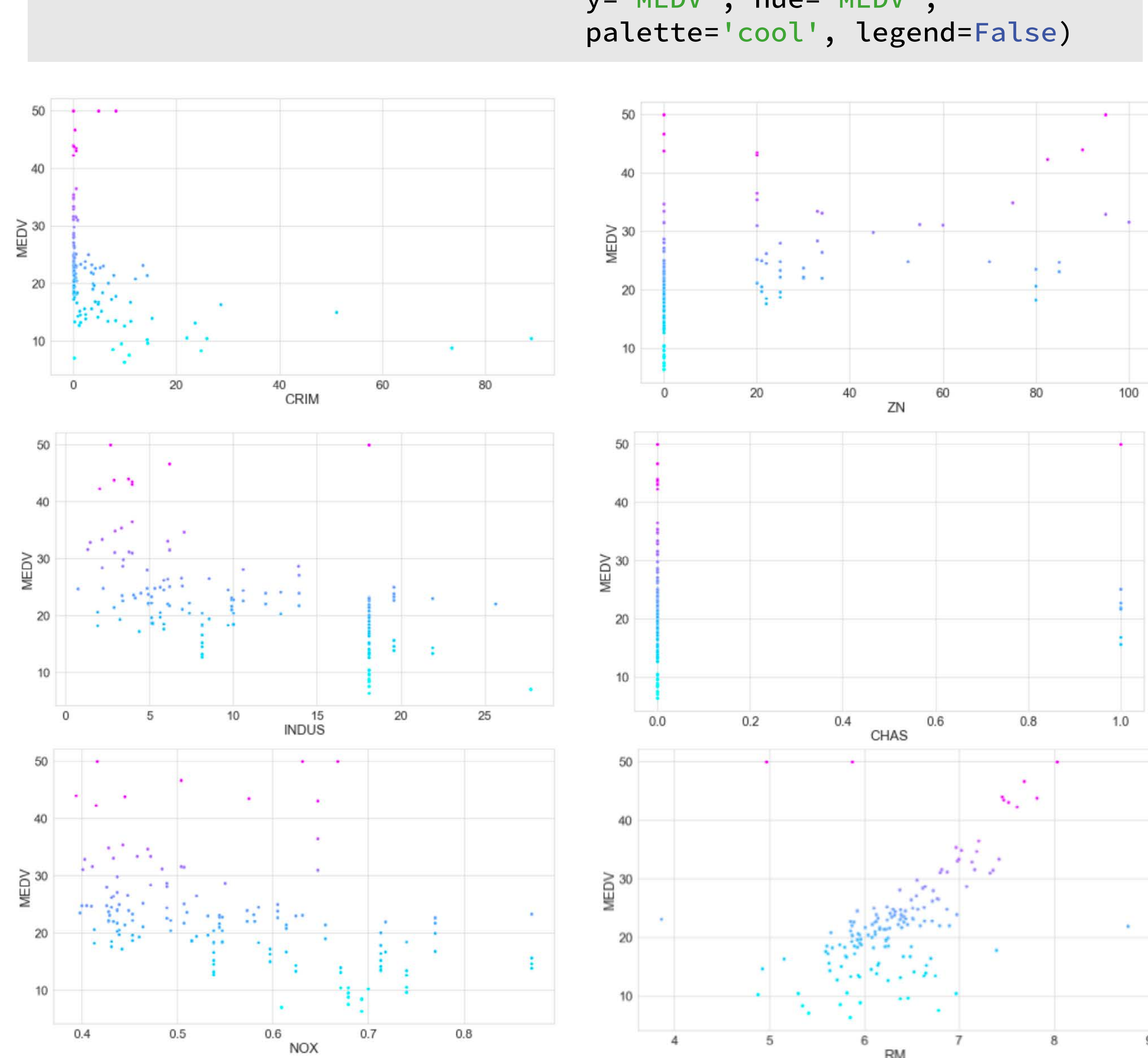
- `CRIM`, tasa de delincuencia per cápita en la ciudad
- `ZN`, proporción de terrenos residenciales divididos en zonas para lotes de más de 25,000 ft .
- `INDUS`, proporción INDUS de acres comerciales no minoristas por ciudad.
- `CHAS`, variable ficticia de Charles River (1 si el tramo limita con el río; 0 en caso contrario)
- `NOX`, concentración de óxido nítrico NOX (partes por 10 millones)
- `RM`, número promedio de habitaciones por vivienda
- `AGE`, proporción de unidades ocupadas por sus propietarios y construidas antes de 1940
- `DIS`, distancias ponderadas DIS a cinco centros de empleo de Boston
- `RAD`, índice de accesibilidad a carreteras radiales
- `TAX`, tasa de impuesto a la propiedad de valor total por $ 10,000

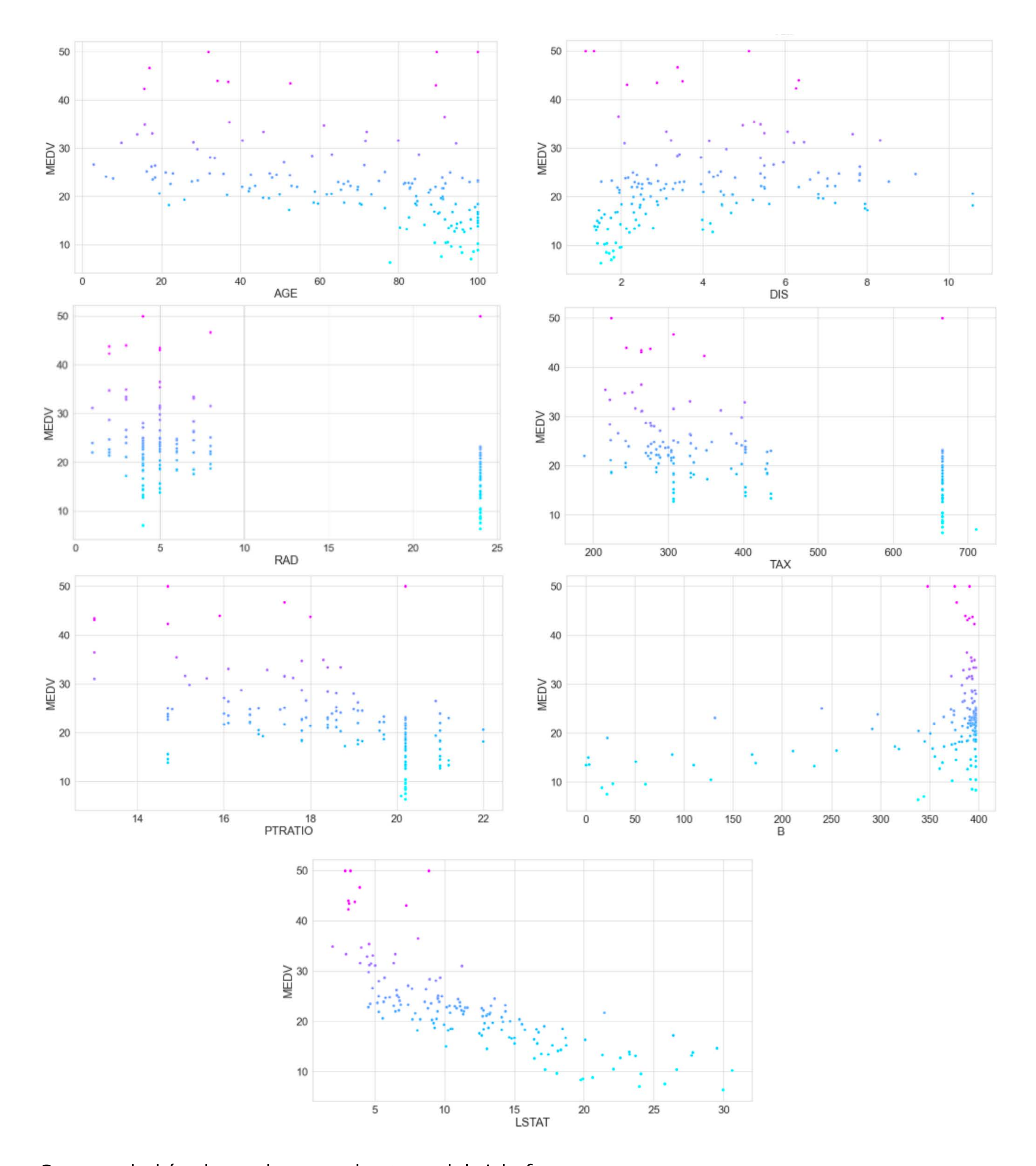
- `PTRATIO`, proporción alumno-profesor por municipio
- `B= 1000`(Bk - 0.63)2 donde Bk es la proporción de negros por ciudad
- `LSTAT`, % status más bajo de la población
- `MEDV`, valor medio de las viviendas ocupadas por sus propietarios en $ 1000

```
[16]: import matplotlib.pyplot as plt
[17]: import seaborn as sns
[18]: sns.set(font_scale=2)
[19]: sns.set_style('whitegrid')
```

Tanto Matplotlib y Seaborn pueden construir los diagramas de dispersión, sin embargo, Seaborn tiene gráficos de mejor calidad y con menos código.

```
[20]: for feature in boston.feature_names:
          plt.figure(figsize=(16, 9))
          sns.scatterplot(data=sample_df, x=feature,
                          y='MEDV', hue='MEDV',
                          palette='cool', legend=False)
```





Como se había planeado, en cada etapa del ciclo for:

- Pyplot genera un gráfico con tamaño de 16x9 in
- Seaborn genera un diagrama de dispersión con la variable (feature) actual en el eje x, y la variable objetivo (boston.target) en el eje vertical.
- Los colores de los puntos están definidos por la variable objetivo (boston.target) a través de hue.

Para realizar la interpretación de los gráficos, siempre será necesario tener a una persona experta en el tema, en este caso, por ejemplo, alguien especializado en bienes raíces.

Sin embargo, algunas pistas, son evidentes al momento de interpretar los diagramas.

- En el primer diagrama, CRIM vs MEDV, tal parece que mientras más pequeño sea el índice de criminalidad el valor promedio de la vivienda será mayor.
- En el diagrama, CHAS vs MEDV, se puede leer que una buena proporción de viviendas colindan con el río y que esta característica no incide en el precio de la vivienda.
- El diagrama, RM vs MEDV, indica que mientras mayor sea el número de habitaciones en las viviendas mayor será el precio de éstas.
- En el diagrama, LSTAT vs MEDV, se puede leer que mientras mayor sea el porcentaje de personas con status bajo, el precio de la vivienda tenderá a disminuir.

15.16 Separación de datos para entrenamiento y prueba

Nuevamente prepararemos los datos para realizar entrenamientos y pruebas.

```
[21]: from sklearn.model_selection import train_test_split
[22]: X_entrena, X_prueba, y_entrena, y_prueba = train_
test_split(boston.data, boston.target, random_state=7)
[23]: X_entrena.shape
[23]: (379, 13)
[24]: X_prueba.shape
[24]: (127, 13)
```

Recuerda que al dejar las opciones por default en import train_test_split, se deja el 75% de los datos para realizar entrenamiento y 25% para realizar pruebas.

15.17 Creación, entrenamiento del modelo

Por default, los estimadores de regresión lineal utilizan a todas las variables del conjunto de datos para realizar la regresión múltiple. Pueden ocurrir errores cuando las variables son categóricas. Por lo que, si existen este tipo de variables, la tarea inicial será transformarlas en variables que si lo sean. Una ventaja de trabajar con los conjuntos de datos de scikit-learn es que ya están preparados para trabajarlos.

Tanto, `X_entrena`, `X_prueba` tienen 13 columnas que representan a las variables independientes.

Para crear un estimador de regresión lineal invocamos el método fit para entrenar al estimador, utilizando `x_entrena`, `y_entrena`.

```
[25]: from sklearn.linear_model import LinearRegression
[26]: regresion_lineal = LinearRegression()
[27]: regresion_lineal.fit(X=X_train, y=y_train)
[27]: LinearRegression()
```

```
[28]: for i, name in enumerate(boston.feature_names):
          print(f'{name:>10}: {regresion_lineal.coef_[i]}')
[28]: CRIM:  -0.12937298631826694
        ZN:   0.02959048702663703
     INDUS:   0.022292842543141126
      CHAS:   2.8374457856309823
       NOX: -15.395420307959458
        RM:   5.2755727349663175
       AGE:  -0.010538384056323387
       DIS:  -1.3017076456601642
       RAD:   0.2663928959347862
       TAX:  -0.010968670237040117
   PTRATIO:  -0.9648301928562035
         B:   0.010860336086222841
     LSTAT:  -0.3783634647357096

[29]: regresion_lineal.intercept_
[29]: 23.95674601756494
```

Estos valores, son estimaciones de los coeficientes desconocidos en el modelo de regresión lineal, es decir

$$\hat{\beta}_0 = 23.95674601756494$$

$$\hat{\beta}_1 = -0.12937298631826694$$

$$\hat{\beta}_2 = 0.02959048702663703$$

$$\vdots$$

$$\hat{\beta}_{12} = 0.010860336086222841$$

$$\hat{\beta}_{13} = -0.3783634647357096$$

Por lo que el modelo tomaría la forma

$$MEDV=\hat{\beta}_0 + \hat{\beta}_1 CRIM + \hat{\beta}_2 ZN + \ldots + \hat{\beta}_{12} B + \hat{\beta}_n LSTAT$$

Para valores específicos de las variables independientes CRIM, ZN, INDUS, etc. Es posible determinar el precio aproximado de una vivienda.

Cuando los coeficientes son de signo positivo, al incrementarse el valor de la variable independiente, el costo de la vivienda también se incrementa, justo como lo habíamos señalado con RM y MEDV. Si el coeficiente es negativo, al incrementarse el valor de la variable independiente, el costo de la vivienda disminuye, como es el caso de LSTAT y MEDV.

15.18 Prueba del modelo

Para verificar que tan bueno es el modelo, puedes utilizar el método predict con las muestras de prueba como argumento.

```
[30]: pronostico = regresion_lineal.predict(X_prueba)
[31]: esperado = y_prueba
```

Podemos ver los primeros 5 pronósticos para darnos una idea de que tan bien está trabajando el modelo:

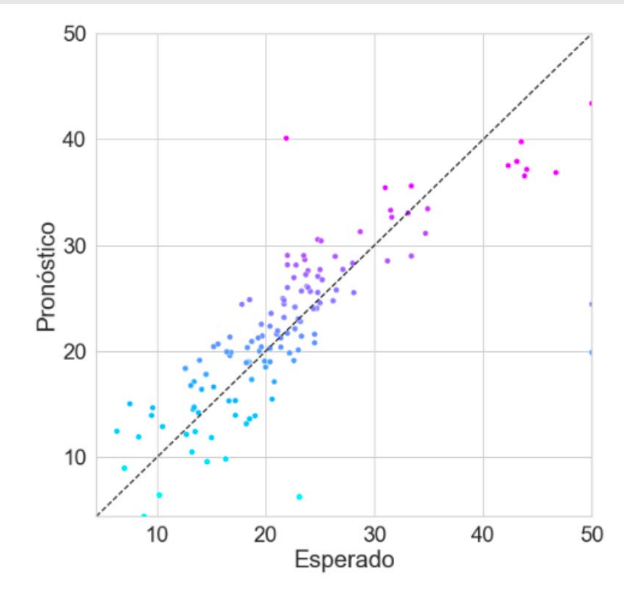
```
[32]: pronostico[:5]
[32]: array([23.1903541, 18.97985889, 19.82548836, 19.00126197,
       4.39524325])
[33]: esperado[:5]
[33]: array([21.7, 18.5, 22.2, 20.4, 8.8])
```

Para visualizar como es la relación entre los precios esperados y los precios pronosticados

```
[34]: df = pd.DataFrame()
[35]: df['Esperado'] = pd.Series(esperado)
[36]: df['Pronóstico'] = pd.Series(pronostico)
```

Ahora se construye el diagrama de dispersión entre los precios esperados y los pronósticos.

```
[37]: figure = plt.figure(figsize=(9, 9))
      axes = sns.scatterplot(data=df, x='Esperado', y='Pronóstico',
      hue='Pronóstico', palette='cool', legend=False)
      inicio = min(esperado.min(), pronostico.min())
      fin = max(esperado.max(), pronostico.max())
      axes.set_xlim(inicio, fin)
      axes.set_ylim(inicio, fin)
      line = plt.plot([inicio, fin], [inicio, fin], 'k--')
```



En una situación ideal, donde los estimadores son iguales a los valores reales, en el modelo, los valores esperados y de pronósticos serían exactamente los mismos. De manera que en la gráfica caerían en una línea $y = x$. En este caso, eso no ocurre, aunque si se alcanza a mostrar un comportamiento adecuado.

15.19 Ajuste y evaluación del modelo

Una de las formas de verificar que tan bueno puede ser un modelo de regresión lineal es el coeficiente de determinación R^2 . Este valor esta definido entre 0 y 1. Si es cercano al cero indica que el modelo construido no es el adecuado para realizar pronósticos. Si es cercano al 1, es un indicador que da mucha confianza al momento de realizar pronósticos.

```
[38]: from sklearn import metrics
[39]: metrics.r2_score(esperado, pronostico)
[40]: 0.6170003090082025
```

Como puedes observar el valor de R^2 es 0.617, que no es tan malo, pero es sugerible buscar otros métodos que puedan mejorar la precisión de los pronósticos.

Otro indicador que muestra que tan bueno es el modelo es el error cuadrático medio. El módulo sklearn.metrics lo puede calcular mediante la función `mean_squared_error`.

```
[41]: metrics.mean_squared_error(esperado, pronostico)
[41]: 29.5151377901978
```

Siendo este indicador una medida del error, es deseable que sea cercano al cero. En este caso, no lo es, pero tampoco representa un valor grande.

15.20 Comparando diferentes modelos

Existen varios métodos que permiten calcular los coeficientes de un modelo de regresión, para realizar una comparación de ellos, podemos utilizar la misma base de datos y verificar el coeficiente de determinación y establecer cuál método proporciona el mejor ajuste.

En este ejemplo utilizaremos los estimadores: linear_regression, ElasticNet, Lasso y Ridge. Para mayor información acerca de estos métodos puedes ingresar a:

https://scikit-learn.org/stable/modules/linear_model.html

El código para esta comparación es el siguiente.

```
[42]: from sklearn.linear_model import ElasticNet, Lasso, Ridge
[43]: estimadores = {
                        'LinearRegression': regresion_lineal,
                        'ElasticNet': ElasticNet(),
                        'Lasso': Lasso(),
                        'Ridge': Ridge()
                        }
```


Una vez más realizaremos una validación cruzada con un objeto KFold y la función `cross_val_score`.

```
[44]: from sklearn.model_selection import KFold, cross_val_score
[45]: for estimator_name, estimator_object in estimadores.items():
          kfold = KFold(n_splits=10, random_state=7, shuffle=True)
          scores = cross_val_score(estimator=estimator_object,
              X=boston.data, y=boston.target, cv=kfold, scoring='r2')
          print(f'{estimator_name:>16}: ' +
                f'mean of r2 scores={scores.mean():.3f}')
[45]: LinearRegression: mean of r2 scores=0.718
            ElasticNet: mean of r2 scores=0.671
                 Lasso: mean of r2 scores=0.662
                 Ridge: mean of r2 scores=0.716
```

De acuerdo al último resultado, los estimadores de LinearRegression y Ridge son los que mejor realizan el ajuste de los datos.

15.20 Machine Learning No Supervisado. Reducción de Dimensiones

La reducción de dimensiones en un problema puede resultar en un gran ahorro de recursos, ya sea en el tiempo de recursos computacionales o en el tiempo invertido en realizar un análisis de datos.

En esta sección aprenderás a utilizar una estrategia sencilla pero poderosa que permite reducir la dimensión de las variables que explican un fenómeno.

Utilizaremos nuevamente la base de datos Digits de sklearn. En este caso no es necesario manipular las etiquetas de las imágenes por lo estarás empleando una máquina de aprendizaje no supervisado.

Primero cargaremos la base de datos:

```
[1]: %matplotlib inline
     from sklearn.datasets import load_digits
[2]: digits = load_digits()
```

Ahora, emplearemos un estimador TSNE, este estimador utiliza el algoritmo t-SNE (del inglés, t-distributed Stochastic Neighbor Embedding) para analizar un conjunto de datos y reducirlos a un número de dimensiones específica.

Crearemos un objeto TSNE para reducir las características de un conjunto de datos a dos dimensiones con el argumento n_components. Como con los otros estimadores dejaremos una semilla generadora fija, con el fin de verificar que ocurren los mismos resultados.

```
[3]: from sklearn.manifold import TSNE
[4]: tsne = TSNE(n_components=2, random_state=7)
```

La reducción de dimensiones con scikit-learn implica dos pasos:

- Entrenar el estimador con el conjunto de datos.
- Utilizar el estimador para transformar a los datos en un número de dimensiones específicas.

En el siguiente snippet aparecen estas dos tareas, al usar el método fit_transform.

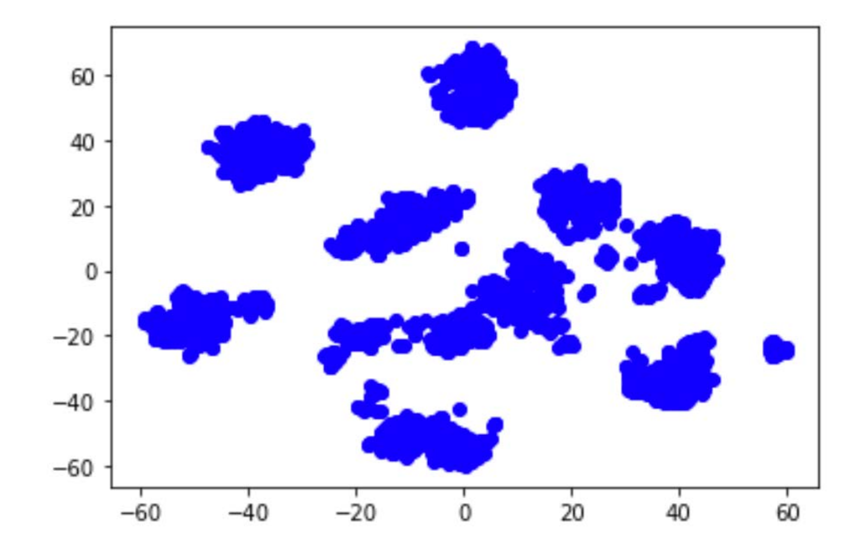
```
[5]: reduced_data = tsne.fit_transform(digits.data)
```

Cuando el método se ejecuta, regresa el mismo número de filas, pero sólo dos columnas.

```
[6]: reduced_data.shape
[6]: (1797, 2)
```

Con la información de las variables reducidas a 2, es posible graficarlas en un plano.

```
[7]: import matplotlib.pyplot as plt
[8]: dots = plt.scatter(reduced_data[:, 0], reduced_data[:, 1],
c='blue')
```

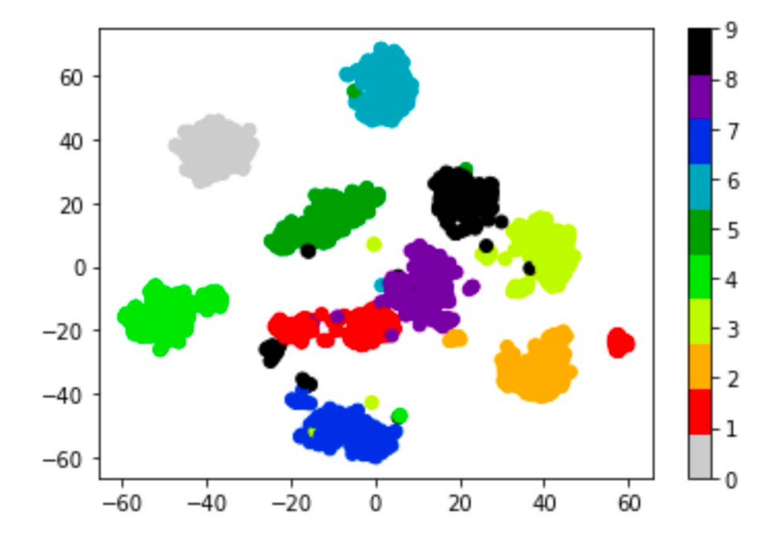


Los primeros dos argumentos de `plt.scatter` son las columnas 0 y 1 con los datos reducidos. El argumento `c='blue'` especifica el color de los puntos. Las nuevas características de los datos reducidos pueden ser muy diferentes a los datos originales.

El diagrama anterior muestra agrupaciones (clusters) de datos relacionados, aparentemente 11, en lugar de los 10 esperados.

Para realizar una mejor visualización, es posible asignar un color a los puntos más cercanos.

```
[7]: dots = plt.scatter(reduced_data[:, 0], reduced_data[:, 1],
c=digits.target, cmap=plt.cm.get_cmap('nipy_spectral_r', 10))
[8]: colorbar = plt.colorbar(dots)
```

Los colores en la paleta del lado derecho muestran la clase a la que representa el color. Esto se realiza a través de `cmap=plt.cm.get_cmap('nipy_spectral_r', 10)`. Observa el cero, de color gris claro está bien separado de los demás ya que en las pruebas anteriores era una de las clases que mejor se podían pronosticar.

Los puntos que no están ubicados con vecinos del mismo color representan aquellos pronósticos erróneos.

CAPÍTULO 16

Aprendizaje Profundo (Deep Learning)

16.1 Introducción

Una de las áreas que ha tenido mayor desarrollo es el llamado Deep learning. Esta es una rama poderosa de machine learning, que ha generado resultados impactantes en diversas y nuevas áreas de las ciencias, como consecuencia del novedoso desarrollo de software y hardware de los últimos años.

Deep Learning ha sido empleado en un rango amplio de aplicaciones

- Reconocimiento facial
- Chatbots
- Visión por computadora
- Conducción automática de vehículos, etc.

16.2 Keras

Los modelos de Deep Learning requieren configuraciones más sofisticadas y regularmente están conectadas a múltiples objetos, llamados capas (layers). En este capítulo construirás modelos con Keras, el cual ofrece una interfaz amigable con TensorFlow de Google.

De acuerdo a TensorFolw.org:

Keras, es la API de alto nivel de TensorFlow para construir y entrenar modelos de aprendizaje profundo. Se utiliza para la creación rápida de prototipos, la investigación de vanguardia (estado-del-arte) y en producción, con tres ventajas clave:

- Amigable al usuario. Keras tiene una interfaz simple y consistente optimizada para casos de uso común. Proporciona información clara y procesable sobre los errores del usuario.
- Modular y configurable. Los modelos en Keras se fabrican conectando bloques de construcción configurables entre sí, con pocas restricciones.
- Fácil de extender. Keras permite escribir bloques de construcción personalizados, para expresar nuevas ideas para la investigación. Crea nuevas capas, métricas, funciones de perdida y desarrolla modelos de estado del arte.
-

Keras es a Deep learning, como Scikit-learn es a machine learning. Como Scikit-learn, Keras mantiene las funciones y los atributos encapsulados, lo cual permite proteger a las funciones originales de modificaciones accidentales.

Los modelos de Deep learning trabajan muy bien para grandes bases de datos, aunque también, puede proporcionar buenos resultados con bases de datos no tan grandes. Estos modelos regularmente realizan una gran cantidad de operaciones, por lo que es recomendable tener un equipo de cómputo con una significativa potencia de procesamiento.

No te preocupes, con un procesador de regular potencia puedes ejecutar los modelos, puede ocurrir que tarden más en ejecutarse, pero en algún instante llegarás al resultado.

Como Scikit-Learn, Keras tiene sus propios conjuntos de datos que resultan útiles para probar modelos. En este capítulo utilizaremos los dos primeros.

- MNIST, esta es una base de datos con dígitos escritos a mano. Esta colección es útil para clasificar imágenes. La base de datos contiene 60,000 imágenes de 28x28 pixeles de los números dígitos, para entrenar y 10,000 para probar.
- IMDb, esta es una base de críticas de cine. Está base de datos permite realizar análisis de sentimientos. Las críticas están etiquetadas como positivas (1) o negativas y cuenta con más de 25,000 críticas para entrenar y 25,000 más para probar.
- Fashion-MNIST, esta es una base de datos de artículos de moda. Es utilizada para clasificar imágenes de 28x28 pixeles de ropa, etiquetadas en 10 categorías con 60,000 muestras para entrenar y 10,000 para probar.
- CIFAR100, es una base de datos para clasificar imágenes pequeñas, contiene 50,000 imágenes a color de 32x32 pixeles etiquetadas en 100 categorías para entrenar y 10,000 imágenes de prueba.

16.3 Instalación de VectorFlow y Keras

Partiendo del hecho de que ya tienes instalado el entorno de Anaconda, realiza los siguientes pasos

Paso 1. Actualiza Anaconda. En el Prompt de Anaconda escribe:

```
conda update conda
```

Luego,

```
conda update –all
```

Paso 2. Actualiza la librería Scikit-learn. En el Prompt de Anaconda escribe:

```
conda update scikit-learn
```

Paso 3. Instala las librerías para Deep Learning En este paso se instalarán las imprescindibles keras y Tensorflow de Google. En el Prompt de Anaconda escribe:

```
conda install -c conda-forge tensorflow
```

Luego,

```
pip install keras
```

Ten paciencia, la actualización de Anaconda y de las librerías de Keras y TensorFlow puede resultar un tanto lentas. Todo depende de la capacidad de tu procesador y de tu red WIFI.

Para dar un vistazo breve a los ejemplos que realizarás, necesitas primero tener algunos antecedentes básicos de Redes Neuronales y de Tensores.

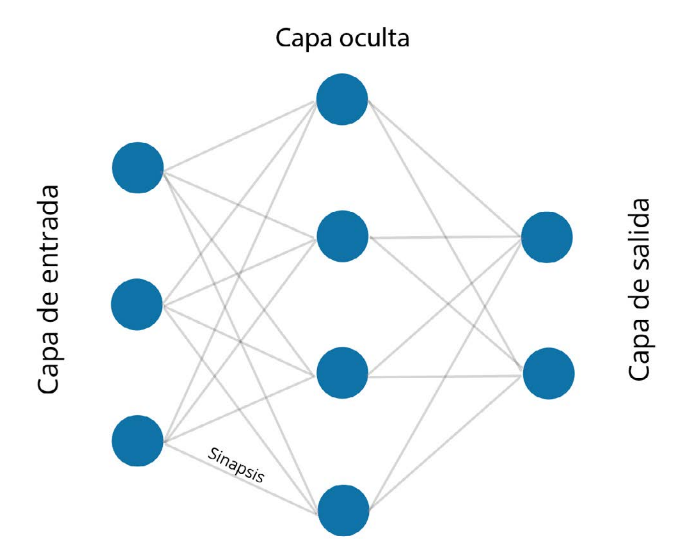
16.4 Redes Neuronales

Una red neuronal artificial o simplemente red neuronal, es un software construido para operar justo como lo hace el cerebro humano. Es decir, las redes neuronales artificiales tratan de imitar la forma en que la gente toma decisiones. O de manera más básica, la forma en que las neuronas biológicas toman decisiones y se comunican con las demás con el fin de aprender.

El siguiente diagrama muestra una red neuronal de 3 capas.

- Cada círculo representa una neurona
- Las líneas entre ellos representan las sinapsis (comunicación).
- La información de salida de una neurona representa la información de entrada de otra

Este diagrama está completamente interconectado, es decir, cada neurona está conectada con otra de la siguiente capa.



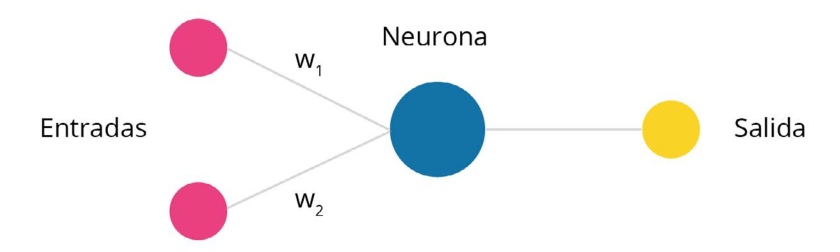
Como el aprendizaje en esta red artificial simula el aprendizaje humano, como en los humanos, la red necesita entrenarse para aprender. Es justo como aprender a programar

- Aprendes a manejar una computadora, el teclado, los atajos.
- Luego utilizas software específico, espero que sea Python.
- Empiezas a utilizar el Prompt de Anaconda, y las herramientas de Jupyter que antes no conocías.
- Luego realizas operaciones sencillas, luego ciclos y sentencias. Y terminas por emplear compresión de listas.

¿Cuándo serás un experto?, cuando hayas entrenado lo suficiente y para diferentes personas el tiempo para aprender fluidamente a programar puede ser diferente. No hay una regla o fórmula que indique el tiempo específico para ser un experto. Esto mismo ocurre con las redes neuronales conforme evolucionan con el tiempo, a cada iteración se le llamaremos epoch (época). De manera que, en cada epoch, la red procesa una muestra del conjunto de datos de entrenamiento a la vez.

Cuando la red se está entrenando calcula algunos coeficientes (como en el caso de regresión), llamados pesos (weigths) para cada conexión entre las neuronas de una capa y las de la siguiente.

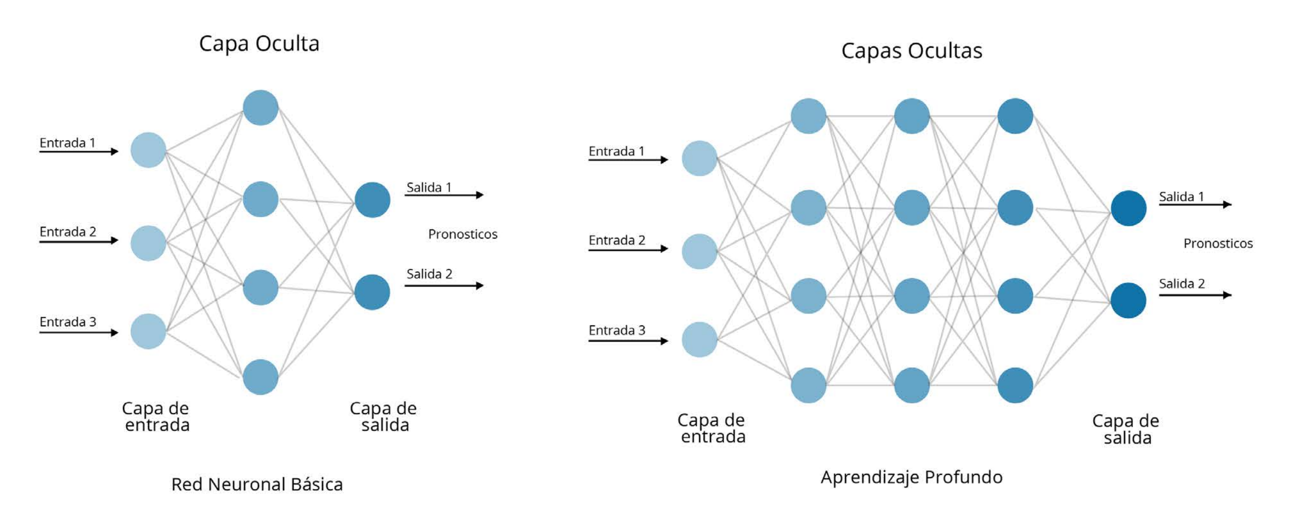
Cuando existe una entrada (input), esta es multiplicada por su respectivo coeficiente y comunicada a la función de activación. Esta función tiene la tarea de activar la neurona o neuronas específicas y dependerá de la entrada que tenga la función. El diagrama siguiente esquematiza este procedimiento.



Los valores w_1, w_2 son llamados pesos o cargas. En un modelo que se entrena desde cero, inicialmente estos valores son seleccionados de manera aleatoria. Cuando la red se entrena, trata de minimizar la tasa de error entre las etiquetas de los pronósticos y las de las muestras. A la tase de error se la llama pérdida. Durante el entrenamiento la red determina la cantidad que cada neurona contribuye a la pérdida total, entonces regresa a la capa y ajusta los pesos con el fin de minimizar tal pérdida.

En resumen, una red neuronal (o modelo) es una secuencia de capas que contienen neuronas utilizadas para aprender de las muestras. En Keras, así es como trabaja una red neuronal básica:

- Cada neurona en la capa, recibe una entrada, las procesa (a través de la función de activación) y produce una salida.
- Los datos se introducen en las redes a través de una capa de entrada, esto define las dimensiones de los datos de muestra.
- Esto es seguido por capas ocultas de neuronas que implementan el aprendizaje y una capa de salida realiza predicciones
- Cuantas más capas se implementen, más profunda será la red. De aquí el término de aprendizaje profundo



16.5 Capas convolucionales

Una capa convolucional utiliza las relaciones entre pixeles vecinos para aprender características útiles (o patrones) en áreas pequeñas de cada muestra. Estas características son entradas de capas subsecuentes.

Las áreas pequeñas, de las que aprende la convolución se llaman kernels o parches. Para entender como funciona la convolución revisemos el siguiente ejemplo. Supongamos que tenemos una imagen de 6x6 y un cuadrado de 3x3 sombreado que representa al kernel. Los números, tanto de la imagen con en el kernel son simplemente para ejemplificar.

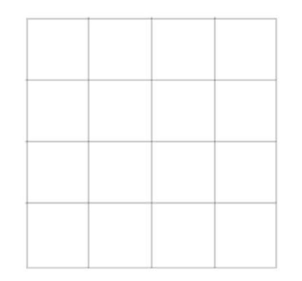
Se puede entender que un kernel es una especie de ventana deslizante y que la convolución mueve la capa convolución un pixel a la vez, primero del extremo superior izquierdo hasta al extremo superior derecho. Y luego baja un pixel y vuelve a realizar el ciclo. Típicamente los kernels son matrices de 3x3, aunque hay versiones de 5x5 y hasta de 7x7 para las imágenes grandes de alta resolución. Después de realizar la convolución por toda la imagen original, queda una matriz de dimensiones más pequeñas que contiene las características de la imagen original.

0	0	0	0	0	0
0	1	1	1	1	1
0	1	1	0	0	0
0	1	1	1	0	0
0	1	0	0	0	0
0	1	1	1	1	0

Entrada de una capa convolucional
Imagen de 6x6
Antes de la convolución

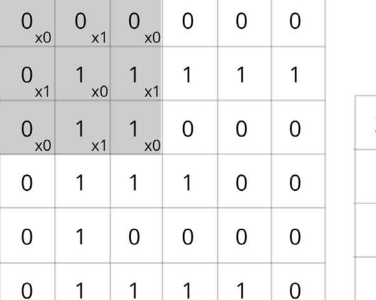
0	1	0
1	0	1
0	1	0

Kernel

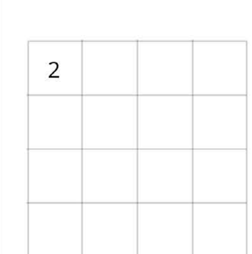


Salida de una capa convolucional
Imagen de 4x4
Después de la convolución

Paso 1



0 x0	0 x1	0 x0	0	0	0
0 x1	1 x0	1 x1	1	1	1
0 x0	1 x1	1 x0	0	0	0
0	1	1	1	0	0
0	1	0	0	0	0
0	1	1	1	1	0



2			

Paso 2

0	0 x0	0 x1	0 x0	0	0
0	1 x1	1 x0	1 x1	1	1
0	1 x0	1 x1	0 x0	0	0
0	1	1	1	0	0
0	1	0	0	0	0
0	1	1	1	1	0

2	3		

Paso 3

0	0	0 x0	0 x1	0 x0	0
0	1	1 x1	1 x0	1 x1	1
0	1	1 x0	0 x1	0 x0	0
0	1	1	1	0	0
0	1	0	0	0	0
0	1	1	1	1	0

2	3	2	

Paso 4

0	0	0	0 x0	0 x1	0 x0
0	1	1	1 x1	1 x0	1 x1
0	1	1	0 x0	0 x1	0 x0
0	1	1	1	0	0
0	1	0	0	0	0
0	1	1	1	1	0

2	3	2	1

Paso 5

0	0	0	0	0	0
0 x0	1 x1	1 x0	1	1	1
0 x1	1 x0	1 x1	0	0	0
0 x0	1 x1	1 x0	1	0	0
0	1	0	0	0	0
0	1	1	1	1	0

2	3	2	1
3			

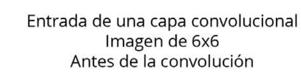
Paso 15

0	0	0	0	0	0
0	1	1	1	1	1
0	1	1	0	0	0
0	1	1	1 x0	0 x1	0 x0
0	1	0	0 x1	0 x0	0 x1
0	1	1	1 x0	1 x1	0 x0

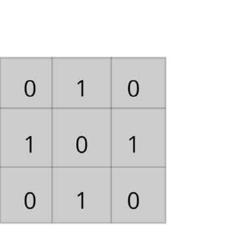
2	3	2	1
3	3	3	1
3	3	1	1
2	3	2	1

La matriz de 4x4 que resulta de cubrir totalmente a la imagen con el kernel se le llama filtro. Esto indica que las dimensiones se redujeron 2 unidades. Entonces, para una imagen de 28x28 se necesita un filtro de 26x26.

0	0	0	0	0	0
0	1	1	1	1	1
0	1	1	0	0	0
0	1	1	1	0	0
0	1	0	0	0	0
0	1	1	1	1	0



Entrada de una capa convolucional
Imagen de 6x6
Antes de la convolución



0	1	0
1	0	1
0	1	0

Kernel

Filtro

2	3	2	1
3	3	3	1
3	3	1	1
2	4	2	1

Salida de una capa convolucional
Imagen de 4x4
Después de la convolución

Al conjunto de filtros producidos mediante la convolución se le llama mapa característico.

Subsecuentemente, las capas de convolución combinan características de mapas característicos previos. Si estuviéramos realizando reconocimiento facial las capas iniciales reconocerían líneas contornos y curvas, la combinación de las capas siguientes proporcionarían otras características como ojos, nariz, oídos y bocas. Una vez que la red a aprendido la característica, la puede reconocer en cualquier lugar de la imagen.

16.6 Tensores

En términos computacionales, un tensor es un arreglo. En términos matemáticos es un vector fila o vector columna de cualquier dimensión. Esta última característica es la que distingue a los tensores, de los vectores habituales de álgebra lineal.

La forma(shape) de un tensor es representada como una tupla de valores en la cual el número de elementos especifican las dimensiones del tensor y cada valor en la tupla especifica el tamaño de la dimensión correspondiente del tensor.

Por ejemplo, supongamos que requieres identificar y rastrear objetos in videos de alta resolución que tienen 30 tramas por segundo. Cada trama en un video 4K de alta resolución tiene 3840x2160 pixeles. Supongamos que los pixeles están representados en un formato RGB (Red, Green, Blue) de color. Así, cada trama tiene un tensor de 3D que contiene 24,883,200 elementos (3840x2160x3) y cada video puede tener un tensor de 4D que contiene una sucesión de tramas. Si el vídeo es de un minuto de duración, entonces hay 44,789,760,000 elementos por tensor.

En Google se cargan aproximadamente 600hrs de video por minuto. Así que Google contiene tensores de 1,612,431,360,000,000 elementos, los cuales requieren un rápido procesamiento cuando se entrenan modelos de aprendizaje profundo.

16.7 Clasificación Múltiple

A continuación, se presenta un ejemplo de clasificación de números dígitos. Aunque este ejemplo es muy parecido al del capítulo anterior, las herramientas y las condiciones que se aplican resultan muy diferentes. Esto es:

- Utilizaremos la base de datos de MNIST la cual tiene 60,000 dígitos etiquetados, para el entrenamiento y 10,000 para realizar pruebas.
- Cada muestra (imagen) es de 28x28 pixeles (784 variables o características)
- Cada pixel toma valores entre 0 y 255, que representan la intensidad en escala de grises de cada pixel.
- El modelo predictor adecuado, en este caso, será el de clasificación probabilística.
- La reproductibilidad de los resultados, en este caso no es posible, ya que las librerías que se emplean realizan operaciones en paralelo, lo cual implica que, al momento de realizar los cálculos, estos se realizan en diferente orden, y los resultados no son idénticos en ejecuciones sucesivas.
- Exploraremos el aprendizaje profundo con una red neuronal convolucional (CNN, por las siglas convolutional neural network) o simplemente convnet.

16.8 Carga de Datos

Para cargar la base de datos desde el módulo de tensorflow

```
[1]: from tensorflow.keras.datasets import mnist En esta
parte se dividen los datos para entrenamiento y pruebas
[2]: (X_entrena, y_entrena), (X_prueba, y_prueba) = mnist.
load_data()
```

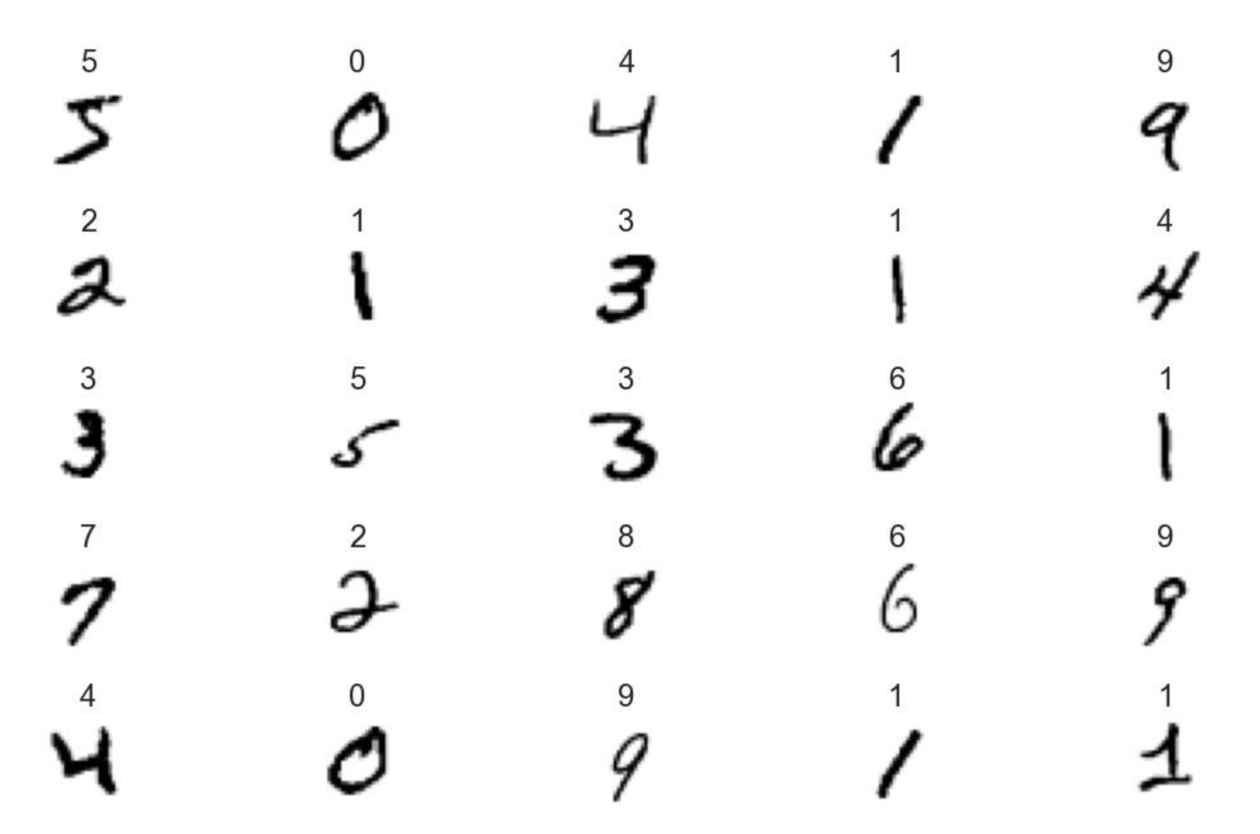
16.9 Exploración de datos

Revisemos las dimensiones del conjunto de imágenes de entrenamiento

```
[3]: X_entrena.shape
[3]: (60000, 28, 28)
[4]: y_entrena.shape
[4]: (60000, )
[5]: X_prueba.shape
[5]: (10000, 28, 28)
[6]: y_prueba.shape
[6]: (10000, )
```

Con esta parte de código puedes visualizar una parte del conjunto de datos

```
[7]:%matplotlib inline
    import matplotlib.pyplot as plt
[8]: import seaborn as sns
[9]: sns.set(font_scale=2) [10]:
[10]: import numpy as np
      index = np.random.choice(np.arange(len(X_entrena)),
      25, replace=False)
[11]: figure, axes = plt.subplots(nrows=5, ncols=5,
      figsize=(16, 9))
[12]: for item in zip(axes.ravel(), X_entrena, y_entrena):
          axes, image, target = item
          axes.imshow(image, cmap=plt.cm.gray_r)
          axes.set_xticks([]) # remove x-axis tick marks
          axes.set_yticks([]) # remove y-axis tick marks
          axes.set_title(target)
      plt.tight_layout()
```



Parte de los inconvenientes a los que tendrá que enfrentarse nuestro estimador son:

- Como en la fila 3, el 5 que aparece puede ser confundido con un 6.
- El número 7 de la tercera fila, puede ser confundido con un 1 o un 9.
- Los números 4, algunas veces aparecerán abiertos, como en las imágenes o cerrados.
- Los números 1, aparecen como una línea simple, pero a veces aparecen con un guión bajo. Este es el caso de los unos de la última fila

Como las imágenes son de 28x28 pixeles, puedes observar que tienen mejor resolución.

16.10 Preparación de los datos

Keras convents requiere un arreglo NumPy de entrada con la sintaxis: (ancho, alto, canales). Para MNIST, cada imagen tienen ancho y alto de 28 pixeles, y cada pixel tiene un canal (la escala de gris del pixel de 0 a 255), por lo que cada muestra debe tener entradas (28,28, 1).

Si las imágenes fueran a colores, los canales serían 3, uno por cada componente RGB.
Para redimensionar las imágenes

```
[13]: X_entrena = X_entrena.reshape((60000, 28, 28, 1))
[14]: X_entrena.shape
[14]: (60000, 28, 28, 1)
[15]: X_prueba = X_prueba.reshape((10000, 28, 28, 1))
[16]: X_prueba.shape
[16]: (10000, 28, 28, 1)
```

Como las características numéricas pueden tener un amplio rango de valores, las redes para Deep learning funcionan mucho mejor cuando los datos están estandarizados o normalizados. Una forma de realizarlo es dividir todos los datos entre el valor más grande, de manera que los datos ahora tendrán un rango entre cero y uno. En código, esto es:

```
[17]: X_entrena = X_entrena.astype('float32') / 255
[18]: X_prueba = X_prueba.astype('float32') / 255
```

El modelo de predicción convent de Keras genera un arreglo de 10 probabilidades, donde la mayor, indica la similitud a la que el dígito debe pertenecer (clase del 0 al 9). Al momento de evaluar la precisión del modelo, Keras compara el pronóstico con la etiqueta, por lo tanto, Keras requiere que el pronóstico y la etiqueta sean de la misma dimensión. Esto provoca un conflicto ya que la probabilidad se presenta en un arreglo y las etiquetas son un entero. Una forma de resolver este problema es convertir las etiquetas en arreglos. A este procedimiento se le llama one-hot encoding.

El módulo tensorflow.keras.utils, puede realizar este procedimiento a través de la función to_categorical, colocando en un arreglo 9 ceros y un 1 en el lugar correspondiente a la etiqueta, de esta forma, la representación categórica del número 8 es

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
```

Y del 2

```
[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Para transformar y_entrena y y_prueba, de arreglos con una dimensión que contienen valores de 0 a 9, en arreglos de dos dimensiones de datos categóricos.

```
[19]: from tensorflow.keras.utils import to_categorical
[20]: y_entrena = to_categorical(y_entrena)
[21]: y_entrena.shape
```



```
[21]: (60000, 10)
[22]: y_entrena[0]
[22]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
      dtype=float32)
[23]: y_prueba = to_categorical(y_prueba)
[24]: y_prueba.shape
[24]: (10000, 10)
```

En el snippet [5] se muestra la forma categorizada del primer elemento en la lista de imágenes que obtuvimos anteriormente (número 5).

Ya todo está preparado para crear una red neuronal convolucional. Iniciaremos de la siguiente manera

```
[25]: from tensorflow.keras.models import Sequential
[26]: cnn = Sequential()
```

El modelo Sequential de Keras permite que la red neuronal ejecute sus capas secuencialmente, esto quiere decir que la salida de una capa es la entrada de la siguiente.

La red neuronal consiste de varias capas, de manera que las capas de entrada reciben las muestras de entrenamiento, las capas ocultas aprenden de las muestras y las capas de salida pueden realizar pronósticos. Para crear una red convolucional básica, importaremos las clases `Conv2D`, `Dense`, `Flatten` y `MaxPooling2D`.

```
[27]: from tensorflow.keras.layers import Conv2D, Dense,
Flatten, MaxPooling2D
```

Añadiremos una capa de convolución al modelo

```
[28]: cnn.add(Conv2D(filters=64, kernel_size=(3, 3), acti-
vation='relu', input_shape=(28, 28, 1)))
```

Añadiremos una capa MaxPooling2D al modelo

```
[29]: cnn.add(MaxPooling2D(pool_size=(2, 2)))
```

Ahora añadiremos na segunda capa de convolución con 128 filtros, seguido de una capa Pooling para reducir la dimensionalidad.

```
[30]: cnn.add(Conv2D(filters=128, kernel_size=(3, 3),
      activation='relu'))
[31]: cnn.add(MaxPooling2D(pool_size=(2, 2)))
```

Hasta este punto la salida de la última capa regresa un arreglo de tres dimensiones (5x5x128), pero la salida final debe un arreglo de una dimensión de 10 probabilidades para clasificar a los números dígitos. Para preparar el pronóstico final utilizaremos la capa Flatten de Keras.

```
[32]: cnn.add(Flatten())
```

La siguiente capa Dense crea 128 neuronas (units) que aprenden de las 3200 salidas de la capa anterior.

```
[33]: cnn.add(Dense(units=128, activation='relu'))
```

La capa final es una capa Dense que clasifica las entradas en neuronas representativas de las clases 0 al 9. La función de activación softmax convierte los valores de estas neuronas en probabilidades para la clasificación.

```
[34]: cnn.add(Dense(units=10, activation='softmax'))
```

El método summary muestra las capas utilizadas en el modelo

```
[35]: cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 64)        640
_______________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 3200)              0
_________________________________________________________________
dense (Dense)                (None, 128)               409728
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 485,514
Trainable params: 485,514
Non-trainable params: 0

_________________________________________________________________
```

Una vez que se han añadido las capas necesarias, es posible realizar una compilación del modelo. Esto se realiza mediante el método compile.

```
[36]: cnn.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```


En este caso

- El optimizador `optimizer='adam'` realiza un ajuste de los pesos a través de la red neuronal. Aunque hay muchos optimizadores, `'adam'` realiza una muy buena validación cruzada en una gran variedad de modelos.
- La función de pérdida `'categorical_crossentropy'`, es utilizada por el optimizador en redes de multiclasificación, recuerda que esn este caso debe realizar pronóstico de 10 clases. Para la clasificación binaria, Keras tiene `'binary_crossentropy'` y para regresión `'mean_squared_error'`. Si requieres más información u otras funciones de perdida puedes consultar: https://keras.io/api/metrics/

16.11 Entrenamiento y Evaluación del Modelo

Para entrenar el modelo:

```
[37]: cnn.fit(X_entrena, y_entrena, epochs=5, batch_size=64, va-
lidation_split=0.1)
[37]:Epoch 1/5
844/844 [==============================] - 126s 149ms/step - loss:
0.0452 - accuracy: 0.9860 - val_loss: 0.0377 - val_accuracy: 0.9895
Epoch 2/5
844/844 [==============================] - 136s 161ms/step - loss:
0.0262 - accuracy: 0.9917 - val_loss: 0.0424 - val_accuracy: 0.9868
Epoch 3/5
844/844 [==============================] - 149s 176ms/step -  loss:
0.0171 - accuracy: 0.9941 - val_loss: 0.0355 - val_accuracy: 0.9915
Epoch 4/5
844/844 [==============================] - 146s 173ms/step -  loss:
0.0126 - accuracy: 0.9959 - val_loss: 0.0332 - val_accuracy: 0.9918
Epoch 5/5
844/844 [==============================] - 147s 174ms/step - loss:
0.0103 - accuracy: 0.9966 - val_loss: 0.0429 - val_accuracy: 0.9898
<tensorflow.python.keras.callbacks.History at 0x1fb55a49c40>
```

Como en el caso de Scikit-Learn

- El modelo se entrena llamando al método fit.
- Los primeros dos argumentos son los datos de entrenamiento y las etiquetas objetivo.
- **epochs**, es el número de veces que el modelo debe procesar el conjunto entero de datos de entrada.
- `batch_size` es el número de muestras a procesar en un intervalo de tiempo durante cada epoch.
- Muchos modelos especifican una potencia de 2, de 32 a 512. En este caso probamos con 64. Un batch demasiado grande puede afectar la precisión del modelo.
- `validation_split=0.1`, es el porcentaje de datos (en este caso 10%) que se dejan para realizar la validación de datos después de cada epoch.

Observa en los resultados que en cada Epoch se han revisado 844 de 844 lotes (batch) que multiplicados por 64 resultan en aproximadamente 54,000 muestras. En la realización de cada epoch el programa ha tardado entre 126 a149 segundos, que puede pensarse como lento, pero está sujeto al procesador que se esté utilizando. Con negritas se han resaltado la precisión del entrenamiento y de la validación cruzada, los valores se encuentran entre 0.98 y 0.99, los cuales son buenos indicadores para el modelo que estamos empleando.

Para revisar la precisión del modelo sobre los datos de prueba, podemos invocar el método evaluate.

```
[38]: perdida, precision = cnn.evaluate(X_prueba, y_prueba)
[38]: 313/313 [====================] - 9s 30ms/step -
      loss: 0.0333 - accuracy: 0.9913
[39]: perdida
[39]: 0.03334169462323189
[40]: precision
[40]: 0.9912999868392944
```

De acuerdo a la información anterior, el modelo convnet es 99.13% preciso cuando se utiliza para reconocer imágenes nuevas. Lo cual es bastante bueno, a pesar haber utilizado algunos valores de los argumentos que se encuentran por default en las funciones.

El método predict del modelo pronostica las clases de las imágenes en el arreglo X_test

[41]: pronostico = cnn.predict(X_prueba)

Para verificar la primera imagen

[42]: y_prueba[0]
[42]: array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)

De acuerdo a esta salida, el número al que pertenece es al 7. Para revisar las probabilidades calculadas por el método predict para la primera muestra.

```
[43]: for index, probability in enumerate(pronostico[0]):
      print(f'{index}: {probability:.10%}')
      0:   0.0000000121%
      1:   0.0000002173%
      2:   0.0000009681%
      3:   0.0001072826%
      4:   0.0000000000%
      5:   0.0000000157%
      6:   0.0000000000%
      7: 99.9998569489%
      8:   0.0000000641%
      9:   0.0000304994%
```

Esta información indica que, en el pronóstico, el número observado debe ser el 7 con casi 100% de certeza.

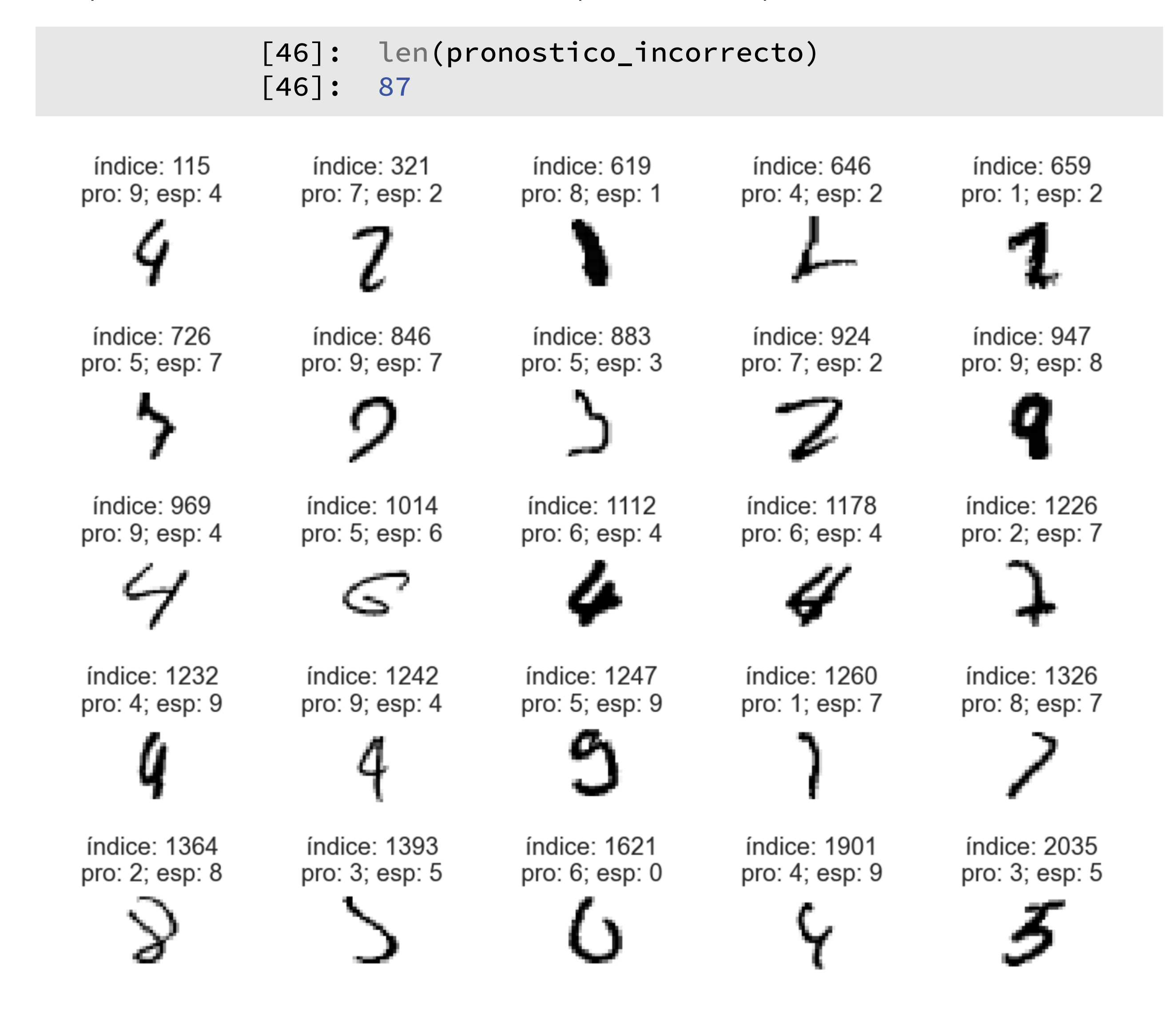
Antes de ver las predicciones incorrectas, es necesario localizarlas. Para realizar esto la función argmax de Numpy determina el índice del elemento más grande en un arreglo, esta característica es bastante útil en este problema, pues permite identificar cuando el pronóstico coincide con el valor esperado.

```
[44]: images = X_prueba.reshape((10000, 28, 28))
      pronostico_incorrecto = []
[45]: for i, (p, e) in enumerate(zip(predictions, y_prueba)):
          pronostico, esperado = np.argmax(p), np.argmax(e)

      if pronostico != esperado:
      pronostico_incorrecto.append((i,  images[i],  pronostico,
      esperado))
```

Ahora podemos ver la cantidad de elementos que fueron mal pronosticados

```
[46]:  len(pronostico_incorrecto)
[46]:  87
```



Para ver algunas imágenes incorrectamente pronosticadas (25)

En cada caso aparece el índice de la imagen, el valor pronosticado pro:, y el valor esperado esp:. Para revisar las probabilidades de algunas muestras incorrectamente clasificadas

```
[44]: def display_probabilities(prediction):
          for index, probability in enumerate(prediction):
              print(f'{index}: {probability:.10%}')
```

Observa las probabilidades de las imágenes con índices 321, 1232, 1326.

```
[45]: display_probabilities(predictions[321])
[46]: 0:  0.0014458373%
      1:  0.0530130579%
      2: 14.2385438085%
      3:  3.8843631744%
      4:  0.0000000008%
      5:  0.0000000034%
      6:  0.0000000029%
      7: 81.6540360451%
      8:  0.1639922732%
      9:  0.0045996054%
```

Este número se esperaba que fuera identificado con el 2, sin embargo, el modelo estima que es 7 con una probabilidad de 81.65%, contra 14.24% de que sea 2.

```
[45]: display_probabilities(predictions[1232])
[46]: 0:  1.9315948710%
      1:  0.3629161976%
      2:  0.0096550219%
      3:  0.0000094327%
      4: 69.7573363781%
      5:  0.0538861670%
      6:  3.1195960939%
      7:  0.0019221816%
      8:  1.6089597717%
      9: 23.1541231275%
```

Este número se esperaba que fuera identificado con el 9, sin embargo, el modelo estima que es 4 con una probabilidad de 69.76%, contra 23.15% de que sea 9. Observa que también hay probabilidades de que sea considerado 6 con 3.12%, 0 con 1.93% y 8 con 1.61%.

```
[45]: display_probabilities(predictions[1326])
[46]: 0: 0.0079155841%
      1: 0.2348761540%
      2: 0.0108659347%
```



```
      3: 0.0705945073%
      4: 0.0000014757%
      5: 0.0000004007%
      6: 0.0000000436%
      7: 12.5625401735%
      8: 45.9975957870%
      9: 41.1156117916%
```

Este número se esperaba que fuera identificado con el 7, sin embargo, el modelo estima que es 8 con una probabilidad de 69.76%, contra 12.56% de que sea 7. Observa que también hay probabilidades de que sea considerado 9 con 41.12%.

Entrenar una red neuronal puede requerir tiempo significativo, por lo que resulta importante guardar el modelo. Además de acceder a un respaldo es posible aplicar el modelo a otro conjunto de datos o diseñar un nuevo modelo que tenga como módulo, el modelo guardado. A esta última tarea se le llama transferencia de aprendizaje. Keras permite guardar el método para su reutilización posterior en un archivo con extensión .h5.

```
[47]: cnn.save('mnist_cnn.h5')
```

Cuando requieras volver a cargar el modelo puedes utilizar

```
from tensorflow.keras.models import load_model
cnn=load_model('mnist_cnn.h5')
```

16.12 Análisis de Emociones con Keras

Keras también permite acceder a la base de datos IMDb. Esta base de datos contiene 25,000 críticas de películas como muestras para entrenar y cuenta con otras 25,000 para realizar pruebas. Las críticas están clasificadas como positivas (1) o negativas (0)

16.13 Carga de Datos

Para cargar la base de datos desde el módulo de tensorflow

```
[1]: from tensorflow.keras.datasets import imdb
```

El módulo imdb de la función load_data regresa los conjuntos para entrenamiento y prueba. En el conjunto de datos hay alrededor de 88,000 palabras únicas. La función load_data permite establecer un número fijo de palabras únicas de interés. En este ejemplo cargaremos las 10,000 palabras más frecuentes que ocurren, esto se hace por las limitaciones computacionales que nos podamos encontrar.

```
[2]: number_of_words = 10000
[3]: (X_entrena, y_etrena), (X_prueba, y_prueba) =
                 imdb.load_data(num_words=number_of_words)
                 Downloading data from https://storage.goo-
```

```
gleapis.com/tensorflow/tf- keras-datasets/imdb.npz
17465344/17464789 [==============================] - 2s 0us/
step
```

16.14 Exploración de Datos

Revisemos las dimensiones del conjunto de muestras de entrenamiento y de prueba.

```
[4]: X_entrena.shape
[4]: (25000, )
[5]: y_entrena.shape
[5]: (25000, ))
[6]: X_prueba.shape
[6]: (25000, )
[7]: y_ prueba.shape
[7]: (25000, ))
```

Los arreglos y_entrena y y_prueba son arreglos unidimensionales que contienen ceros y unos, indicando cuando la crítica fue positiva o negativa. Sin embargo X_entrena y X_prueba también aparecen como arreglos unidimensionales, peor sus elementos son una lista de enteros cada uno representando los contenidos de una crítica. Aquí una muestra.

```
[8]: %print
     Pretty printing has been turned OFF
[9]: X_train[123]
[9]: array([ 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    1,    5,   14,
            9,    6,   55, 1193,   22,   13,  203,   30,  355,   21,   14,
            9,    4,  236,   22,  121,   13, 1192, 2967, 3622,    5,  779,
          284,   37, 5844,    4,  217,    5, 2132,    6,  749,   10,   10,
         2636, 4252,    5, 2931, 4517,   26,   82,  321,   36,   26,    2,
            5, 4960,    2, 1786,    8,  358,    4,  704,  117,  122,   36,
          124,   51,   62,  593,  375,   10,   10,    4, 1381,    5,  732,
           26,  821,    5, 1249,   14,   16,  159,    4,  504,    7, 3728,
         4913,   10,   10,   51,    9,   91, 1193,   44,   14,   22,    9,
            4,  192,   15, 1370,   40,   14,  131, 1778,   11,  938,  704,
         3834,  131,    2,  543,   84,   12,    9,  220,    6, 1117,    5,
            6,  320,  237,    4, 3286,  325,   10,   10,   25,   80,  358,
           14,   22,   12,   16,  814,   11,    4, 3968, 8084,    7, 1226,
         7111,   63,  131, 1778,   43,   92, 1278,  501,   15,    8, 6353,
            2,   15, 1609,  131,   47,   24,   77,    2,  237,    2,    2,
          158,  158])
```

Los modelos de aprendizaje profundo de Keras requieren datos numéricos, sin embargo, las críticas se encuentran codificadas. Para ver el texto original es necesario conocer la palabra a la cual

corresponde el valor. Los valores de las palabras corresponden a un ranking, por ejemplo, la palabra numerada 1 es la palabra que más veces aparece en las críticas. La palabra numerada como 2 es la segunda que más aparece y así sucesivamente.

Aunque el diccionario de valores de Keras el 1 es para la palabra más frecuente, los valores 0,1 y 2 tienen los siguientes propósitos:

- El valor 0 en una crítica representa el padding (relleno). Los algoritmos de Keras necesitan que las muestras tengan las mismas dimensiones, por lo que algunas críticas pueden necesitar expandirse a una longitud específica. Esta expansión es llamada relleno, y se rellena con ceros.
- El valor 1 representa el token, este es utilizado internamente para indicar el inicio de una secuencia de texto.
- El valor 2 representa una palabra desconocida. Esto puede ocurrir al limitar el número de datos cargados.

Como ejemplo, decodificaremos una crítica

```
[10]: word_to_index = imdb.get_word_index()
Downloading data from https://storage.googleapis.com/ten-
sorflow/tf- keras-datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/
step
```

La palabra 'amazing' puede aparecer en una crítica positiva, por lo que es interesante saber si aparece en el diccionario

```
[11]:  word_to_index['amazing']
[11]:  447
```

De acuerdo a este valor. La palabra 'amazing' en el conjunto de datos es la 477 palabra más frecuente

Para transformar el rating numérico en palabras, primero invertiremos el orden del word_to_index en el diccionario, y luego una entrada en el nuevo diccionario con la expresión index:word

La siguiente compresión de lista muestra las primeras 50 palabras del nuevo diccionario. En este caso el índice 1 es la que más veces se repite.

```
[12]: index_to_word = \
{index: word for (word, index) in word_to_index.items()}
[13]: [index_to_word[i] for i in range(1, 51)]
[13]: ['the', 'and', 'a', 'of', 'to', 'is', 'br', 'in',
'it', 'i', 'this', 'that', 'was', 'as', 'for', 'with',
'movie', 'but', 'film', 'on', 'not', 'you', 'are', 'his',
'have', 'he', 'be', 'one', 'all', 'at', 'by', 'an', 'they',
'who', 'so', 'from', 'like', 'her', 'or', 'just', 'about',
"it's", 'out', 'has', 'if', 'some', 'there', 'what', 'good',
'more']
```

Observa que muchas de estas palabras son stop words. Tal vez en alguna aplicación sea necesario remover estas palabras.

Ahora decodificaremos una crítica:

```
[14]: ' '.join([index_to_word.get(i - 3, '?') for i in X_
train[150]])
[14]: "?????????????????????????????????? ? ? ? ? ? ? ? ?
and this is a very disturbing film i may be wrong but this
is the last film where i considered burt reynolds an actual
actor who transformed the role and delivered a message br
br jon voight and ned beatty are also excellent they are ?
and unaware ? wanting to enjoy the country little did they
know what would happen next br br the photography and sets
are realistic and natural this was before the days of wes
craven br br what is most disturbing about this film is the
fact that places like this still exist in america country
folk still ? city people it is almost a century and a half
since the civil war br br you will enjoy this film it was
filmed in the rural sections of south georgia which still
exist just don't drive past that to mobile ? that area still
has not been ? since ? ? 10 10"
```

Para verificar de que tipo es la crítica

```
[15]:  y_train[150]
[15]:  1
```

Por lo tanto, es positiva.

16.15 Preparación de los datos

Keras requiere que todas las muestras tengan el mismo tamaño, por lo que restringiremos el número de palabras de las críticas. Aunque, puede haber críticas que necesiten rellenarse. La función pad_sequences del módulo tensorflow.keras.preprocessing.sequence puede realizar esta tarea.

```
[16]: words_per_review = 200
```

Aquí limitaremos el número de palabras por crítica

```
[17]: from tensorflow.keras.preprocessing.sequence
      import pad_sequences
[18]: X_train = pad_sequences(X_train, maxlen=words_per_review)
[19]: X_train.shape
[19]: (25000, 200)
```


En este convnet, utilizaremos el argumento del módulo fit para indicar que 10% de los datos de entrenamiento debe reservarse para validar el modelo a medida que se entrena. En este ejemplo dividiremos las 25000 muestras en 20000 muestra de prueba y 5000 muestras para validación.

```
[20]: from sklearn.model_selection import train_test_split
[21]:  X_test, X_val, y_test, y_val = train_test_split(X_
test, y_test, random_state=11, test_size=0.20)
```

Podemos revisar que las dimensiones estén en orden.

```
[22]: X_prueba.shape
[22]: (20000, 200)
[23]: X_val.shape
[23]: (5000, 200))
```

16.16 Creación de la red neuronal

Configuremos la RNN. Iniciaremos con un modelo secuencial y añadiremos una capa

[24]: from tensorflow.keras.models import Sequential
[25]: rnn = Sequential()

Ahora, es necesario importar las capas que utilizaremos en el modelo.

[26]: from tensorflow.keras.layers import Dense, LSTM
[27]: from tensorflow.keras.layers import Embedding

Para reducir la dimensionalidad, la RNN inicia con una capa embebida, que codifica cada palabra en una más compacta llamada representación denso-vector. Los vectores producidos por la capa embebida guardan el contexto de la palabra, es decir, como una palabra se relaciona con otras a su alrededor. De esta manera la capa embebida habilita a la RNN para aprender relaciones entre palabras durante el entrenamiento.

Para crear una capa embebida

```
[28]: rnn.add(Embedding(input_dim=number_of_words, output_dim=128,
                        input_length=words_per_review))
```

Donde

- Input_dim, es el número de palabras únicas
- Output_dim, es el tamaño de cada palabra embebida
- input_length=words_per_review, es el número de palabras ene cada muestra de entrada.

Ahora añadiremos una capa LSTM

```
[29]: rnn.add(LSTM(units=128, dropout=0.2, recurrent_dropout=0.2))
```

En este caso, los argumentos son:

- units, el número de neuronas en la capa. Mientras más neuronas, más puede recordar la red.
- Dropout, es el porcentaje de neuronas que aleatoriamente se deshabilitan al procesar la entrada y la salida de las capas.
- recurrent_dropout, es el porcentaje de neuronas que aleatoriamente se deshabilitan cuando la salida de las capas está retroalimentando a la capa nuevamente.

Finalmente, es necesario tomar una capa LSTM para reducir todo a un resultado que indique si la crítica es positiva o negativa

```
[30]: rnn.add(Dense(units=1, activation='sigmoid'))
```

aquí la función de activación sigmoid es la que permite lleva a cabo la clasificación binaria.

Ahora es necesario compilar el modelo, por lo que utilizaremos la función 'binary_crossentropy'

```
[31]: rnn.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

El siguiente es un resumen del modelo propuesto

```
[32]: rnn.summary()
(Model: "sequential"

_________________________________________________________________
Layer (type)            Output Shape              Param #
=================================================================
embedding (Embedding)   (None, 200, 128)          1280000

_________________________________________________________________
lstm (LSTM)             (None, 128)               131584

_________________________________________________________________
dense (Dense)           (None, 1)                 129
=================================================================
Total params: 1,411,713
Trainable params: 1,411,713
Non-trainable params: 0

_________________________________________________________________
```

Ahora entrenaremos el modelo.

```
[33]:rnn.fit(X_train, y_train, epochs=5, batch_size=32,
validation_data=(X_val, y_val))
Epoch 1/5
```



```
782/782 [==============================] - 560s 716ms/step - loss:
0.4438 - accuracy: 0.7918 - val_loss: 0.3320 - val_accuracy: 0.8634
Epoch 2/5
782/782 [==============================] - 648s 829ms/step - loss:
0.3231 - accuracy: 0.8661 - val_loss: 0.3360 - val_accuracy: 0.8654
Epoch 3/5
782/782 [==============================] - 532s 680ms/step - loss:
0.2070 - accuracy: 0.9204 - val_loss: 0.3317 - val_accuracy: 0.8682
Epoch 4/5
782/782 [==============================] - 528s 675ms/step - loss:
0.1640 - accuracy: 0.9381 - val_loss: 0.3920 - val_accuracy: 0.8630
Epoch 5/5
782/782 [==============================] - 522s 668ms/step - loss:
0.1276 - accuracy: 0.9527 - val_loss: 0.4512 - val_accuracy: 0.8102
<tensorflow.python.keras.callbacks.History        object        at
0x00000181A4047D 90>
```

Finalmente es posible evaluar al modelo utilizando los datos de prueba

```
[33]: results = rnn.evaluate(X_test, y_test)
625/625 [==============================] - 53s 85ms/step -
loss: 0.45 92 - accuracy: 0.8049
```

En este caso el modelo tiene un 85.99% de precisión, el cual no es malo, pero que puede mejorarse cambiando algunos de los parámetros.

La precisión no es mala si reconocemos que el tipo de problema es mucho más complicado que tratar de interpretar imágenes.